

---

# Table of Contents

Introduction	1.1
Dynamic Programming	1.2
BackPack	1.2.1
Backpack I 92	1.2.1.1
Backpack II 125	1.2.1.2
Backpack III 440	1.2.1.3
Backpack IV 562	1.2.1.4
Backpack V 563	1.2.1.5
Backpack VI 564	1.2.1.6
House Robber 392	1.2.2
House Robber II 534	1.2.3
House Robber III 535	1.2.4
Paint Fence 514	1.2.5
Paint House 515	1.2.6
Paint House II 516	1.2.7
Coins in a Line 394	1.2.8
Coins in a Line II 395	1.2.9
Coins in a Line III 396	1.2.10
Longest Increasing Continuous Subsequence 397	1.2.11
Longest Increasing Continuous subsequence II 398	1.2.12
Maximal Square	1.2.13
Maximal Rectangle 510	1.2.14
Unique Binary Search Trees 163	1.2.15
Unique Binary Search Trees II 164	1.2.16
Unique Paths 114	1.2.17
Unique Paths II 115	1.2.18
Build Post Office 573	1.2.19

---

Build Post Office II 574	1.2.20
Post Office Problem 435	1.2.21
Perfect Squares 513	1.2.22
Longest Palindromic Substring 200	1.2.23
Stone Game 476	1.2.24
Bomb Enemy 553	1.2.25
Burst Balloons 168	1.2.26
Scramble String 430	1.2.27
Copy Books 437	1.2.28
Maximum Gap 400	1.2.29
Longest Common Subsequence 77	1.2.30
Interleaving String 29	1.2.31
Edit Distance 119	1.2.32
Distinct Subsequences 118	1.2.33
Dices Sum 20	1.2.34
Maximum Subarray III 43	1.2.35
Best Time to Buy and Sell Stock IV 393	1.2.36
k Sum 89	1.2.37
Minimum Adjustment Cost 91	1.2.38
Binary Tree Maximum Path Sum 94	1.2.39
Word Break 107	1.2.40
Palindrome Partitioning II 108	1.2.41
Triangle 109	1.2.42
Minimum Path Sum 110	1.2.43
Climbing Stairs 111	1.2.44
Jump Game 116	1.2.45
Jump Game II 117	1.2.46
Maximum Product Subarray 191	1.2.47
Wildcard Matching 192	1.2.48
Decode Ways 512	1.2.49

---

---

Regular Expression Matching 154	1.2.50
Longest Valid Parentheses 32(LeetCode)	1.2.51
Search\Recursion\Backtracking	1.3
Divide and Conquer	1.3.1
Fast Power 140	1.3.1.1
Median of two Sorted Arrays 65	1.3.1.2
Maximum Depth of Binary Tree 97	1.3.1.3
Balanced Binary Tree 93	1.3.1.4
Pow(x, n) 428	1.3.1.5
Find Minimum in Rotated Sorted Array II 160	1.3.1.6
Merge k Sorted Lists 104	1.3.1.7
DFS	1.3.2
Nested List Weight Sum 551	1.3.2.1
Route Between Two Nodes in Graph 176	1.3.2.2
Recursion	1.3.3
Print Numbers by Recursion 371	1.3.3.1
Restore IP Addresses 426	1.3.3.2
Generate Parentheses 427	1.3.3.3
Letter Combinations of a Phone Number 425	1.3.3.4
Backtracking	1.3.4
Word Search 123	1.3.4.1
Word Ladder II 121	1.3.4.2
Combination Sum 135	1.3.4.3
Combination Sum II 153	1.3.4.4
Combinations 152	1.3.4.5
Binary Search	1.3.5
Search in Rotated Sorted Array 62	1.3.5.1
Search in Rotated Sorted Array II 63	1.3.5.2
Search for a Range 61	1.3.5.3
Search Insert Position 60	1.3.5.4

---

---

BFS	1.3.6
Word Ladder 120	1.3.6.1
Number of Islands 433	1.3.6.2
Data Structure	1.4
Tree	1.4.1
Heap	1.4.1.1
Top k largest numbers II 545	1.4.1.1.1
Kth Smallest Number in Sorted Matrix 401	1.4.1.1.2
Super Ugly Number 518	1.4.1.1.3
Kth Largest in N Arrays 543	1.4.1.1.4
Kth Smallest Sum In Two Sorted Arrays 465	1.4.1.1.5
Trapping Rain Water II 364	1.4.1.1.6
Building Outline 131	1.4.1.1.7
Sliding Window Median 360	1.4.1.1.8
Priority Queue	1.4.1.2
Ugly Number II 4	1.4.1.2.1
Data Stream Median 81	1.4.1.2.2
Trie	1.4.1.3
Word Search II 132	1.4.1.3.1
Implement Trie 442	1.4.1.3.2
Add and Search Word 473	1.4.1.3.3
Binary Tree	1.4.1.4
Subtree 245	1.4.1.4.1
Flatten Binary Tree to Linked List 453	1.4.1.4.2
Binary Tree Paths 480	1.4.1.4.3
Binary Tree Maximum Path Sum II 475	1.4.1.4.4
Union Find	1.4.1.5
Graph Valid Tree 178	1.4.1.5.1
Surrounded Regions 477	1.4.1.5.2
Number of Islands 433	1.4.1.5.3

---

---

Find the Connected Component in the Undirected Graph 431	
Find the Weak Connected Component in the Directed Graph 432	1.4.1.5.4 1.4.1.5.5
Number of Islands II 434	1.4.1.5.6
Binary Search Tree	1.4.1.6
Validate Binary Search Tree 95	1.4.1.6.1
Insert Node in a Binary Search Tree 85	1.4.1.6.2
Search Range in Binary Search Tree 11	1.4.1.6.3
Binary Search Tree Iterator 86	1.4.1.6.4
Remove Node in Binary Search Tree 87	1.4.1.6.5
Inorder Successor in BST	1.4.1.6.6
Segment Tree	1.4.1.7
Segment Tree Build 201	1.4.1.7.1
Segment Tree Build II 439	1.4.1.7.2
Segment Tree Query 202	1.4.1.7.3
Segment Tree Query II 247	1.4.1.7.4
Segment Tree Modify 203	1.4.1.7.5
Interval Minimum Number 205	1.4.1.7.6
Interval Sum 206	1.4.1.7.7
Interval Sum II 207	1.4.1.7.8
Count of Smaller Number before itself 249	1.4.1.7.9
Linear	1.4.2
Array	1.4.2.1
Median 80	1.4.2.1.1
Remove Duplicates from Sorted Array 100	1.4.2.1.2
Merge Intervals 156	1.4.2.1.3
First Missing Positive 189	1.4.2.1.4
Partition Array by Odd and Even 373	1.4.2.1.5
Trapping Rain Water 363	1.4.2.1.6
Move Zeroes 539	1.4.2.1.7
Spiral Matrix 374	1.4.2.1.8

---

---

Spiral Matrix II 381	1.4.2.1.9
Plus One 407	1.4.2.1.10
Container With Most Water 383	1.4.2.1.11
The Smallest Difference 387	1.4.2.1.12
Permutation Sequence 388	1.4.2.1.13
Number of Airplanes in the Sky 391	1.4.2.1.14
Minimum Size Subarray Sum 406	1.4.2.1.15
Wiggle Sort 508	1.4.2.1.16
Reverse Pairs 532	1.4.2.1.17
Load Balancer 526	1.4.2.1.18
Candy 412	1.4.2.1.19
Merge k sorted Arrays 486	1.4.2.1.20
Continuous Subarray Sum 402	1.4.2.1.21
Continuous Subarray Sum II 403	1.4.2.1.22
Largest Rectangle in Histogram 122	1.4.2.1.23
Rotate Array (LeetCode) 189	1.4.2.1.24
Linked List	1.4.2.2
Swap Two Nodes in Linked List 511	1.4.2.2.1
Intersection of Two Linked Lists 380	1.4.2.2.2
Reverse Nodes in k-Group 450	1.4.2.2.3
Stack	1.4.2.3
Flatten Nested List Iterator 528	1.4.2.3.1
Valid Parentheses 423	1.4.2.3.2
Evaluate Reverse Polish Notation 424	1.4.2.3.3
Simplify Path 421	1.4.2.3.4
Max Tree 126	1.4.2.3.5
Expression Tree Build 367	1.4.2.3.6
Mock Hanoi Tower by Stacks 227	1.4.2.3.7
Hash Table	1.4.2.4
Happy Number 488	1.4.2.4.1

---

Intersection of Two Arrays 547	1.4.2.4.2
Intersection of Two Arrays II 548	1.4.2.4.3
Minimum Window Substring 32	1.4.2.4.4
Max Points on a Line 186	1.4.2.4.5
Longest Substring Without Repeating Characters 384	1.4.2.4.6
Substring with Concatenation of All Words 30(LeetCode)	
Group Anagrams 49 ( LeetCode )	1.4.2.4.8 1.4.2.4.7
Contains Duplicate (LeetCode) 217	1.4.2.4.9
Contains Duplicate II (LetCode) 219	1.4.2.4.10
Contains Duplicate III (LeetCode) 220	1.4.2.4.11
Deque	1.4.2.5
Sliding Window Maximum 362	1.4.2.5.1
Matrix	1.4.2.6
Matrix Zigzag Traversal 185	1.4.2.6.1
Valid Sudoku 389	1.4.2.6.2
Rotate Image 161	1.4.2.6.3
Search a 2D Matrix II 38	1.4.2.6.4
Set Matrix Zeroes 162	1.4.2.6.5
Sliding Window Matrix Maximum 558	1.4.2.6.6
Submatrix Sum 405	1.4.2.6.7
Find Peak Element II 390	1.4.2.6.8
Sudoku Solver 37 (LeetCode)	1.4.2.6.9
Max Tree 126	1.4.3
Top K Frequent Words 471	1.4.4
Implement Stack 495	1.4.5
Implement stack by two queues 494	1.4.6
Implement Queue by Linked List II 493	1.4.7
Implement Queue by Linked List 492	1.4.8
Stack Sorting 229	1.4.9
Animal Shelter 230	1.4.10

---

LFU Cache 24	1.4.11
LRU Cache 134	1.4.12
Graph & Search	1.5
Clone Graph 137	1.5.1
Copy List with Random Pointer 105	1.5.2
Topological Sorting 127	1.5.3
Permutations 15	1.5.4
Permutations 16	1.5.5
Subsets 17	1.5.6
Subsets II 18	1.5.7
N-Queens 33	1.5.8
N-Queens II 34	1.5.9
Palindrome Partitioning 136	1.5.10
String Permutation II 10	1.5.11
k Sum II 90	1.5.12
Two Pointers	1.6
Two Sum II 443	1.6.1
Remove Element 172	1.6.2
Triangle Count 382	1.6.3
Valid Palindrome 415	1.6.4
Linked List Cycle II 103	1.6.5
Longest Substring with At Most K Distinct Characters 386	1.6.6
Subarray Sum II 404	1.6.7
4 Sum	1.6.8
Remove Nth Node From End of List 174	1.6.9
Sort	1.7
Quick Sort	1.7.1
Nuts & Bolts Problem 399	1.7.1.1
Kth Largest Element 5	1.7.1.2
Wiggle Sort II 507	1.7.1.3

---



---

Largest Number 184	1.7.2
Sort Integers II 464	1.7.3
Bit Manipulation	1.8
A + B Problem 1	1.8.1
Count 1 in Binary 365	1.8.2
O(1) Check Power of 2 142	1.8.3
Flip Bits 181	1.8.4
Update Bits 179	1.8.5
Binary Representation 180	1.8.6
Divide Two Integers 414	1.8.7
Gray Code 411	1.8.8
Bitwise AND of Numbers Range (LeetCode) 201	1.8.9
Permutation	1.9
Permutation Index 197	1.9.1
Permutation Index II 198	1.9.2
Next Permutation 52	1.9.3
Next Permutation II 190	1.9.4
Previous Permutation 51	1.9.5
Greedy	1.10
Delete Digits 182	1.10.1
Find the Missing Number 196	1.10.2
Gas Station 187	1.10.3
String	1.11
Roman to Integer 419	1.11.1
Integer to Roman 418	1.11.2
Length of Last Word 422	1.11.3
Space Replacement 212	1.11.4
Two Strings Are Anagrams 158	1.11.5
String to Integer (Leetcode 8)	1.11.6
strStr 13	1.11.7

---

---

Multiply Strings 43 (LeetCode)	1.11.8
Enumeration	1.12
Digit Counts 3	1.12.1
Longest Common Prefix 78	1.12.2
Longest Words 133	1.12.3
Fizz Buzz 9	1.12.4
Mathematics	1.13
Ugly Number 517	1.13.1
ZigZag Conversion (LeetCode 6)	1.13.2
Palindrome Number (Leetcode 9)	1.13.3
Count Primes (LeetCode) 204	1.13.4
Rectangle Area (LeetCode) 223	1.13.5
Others	1.14
Google	1.14.1
Zigzag Iterator 540	1.14.1.1
Zigzag Iterator II 541	1.14.1.2
Insert Interval 30	1.14.1.3
FaceBook	1.14.2
Add Binary 408	1.14.2.1
Count and Say 420	1.14.2.2
No Tag	1.14.3
Left Pad 524	1.14.3.1
Reverse Integer 413	1.14.3.2
Cosine Similarity 445	1.14.3.3
Word Count (Map Reduce) 499	1.14.3.4
Product of Array Exclude Itself 50	1.14.3.5
Shape Factory 497	1.14.3.6
Toy Factory 496	1.14.3.7
Singleton 204	1.14.3.8
其他问题	1.15

---

---

Use Dynamic Circulate Array to Implement Deque	1.15.1
Random Generator	1.15.2
Array Monotonic	1.15.3
待解决	1.16

---

# LintCode题解

记录刷题过程，总结题目类型，归纳方法。

# Dynamic Programming

# BackPack

# Backpack I 92

## Question

Given  $n$  items with size  $A_i$ , an integer  $m$  denotes the size of a backpack. How full you can fill this backpack?

Notice

You can not divide any item into small pieces.

Example

If we have 4 items with size  $[2, 3, 5, 7]$ , the backpack size is 11, we can select  $[2, 3, 5]$ , so that the max size we can fill this backpack is 10. If the backpack size is 12, we can select  $[2, 3, 7]$  so that we can fulfill the backpack.

Your function should return the max size we can fill in the given backpack.

Challenge

$O(n \times m)$  time and  $O(m)$  memory.

$O(n \times m)$  memory is also acceptable if you do not know how to optimize memory.

## Solution

基础背包类问题，用DP解。两种方法本质都一样，只是表述不同。

第一种方法为  $dp[i][j]$  表示  $i$  容量的背包在  $j$  件物品中能取的最大值。分两种情况：一种情况为不要第  $j$  件物品，则最优情况为背包容量为  $i$  时前  $j-1$  件物品的最优值；第二种情况为要第  $j$  件物品，则最优情况为背包容量为  $i$  - 第  $j$  件物品体积时前  $j-1$  件物品的最优值加上第  $j$  件物品的值，这种情况要求  $i$  - 第  $j$  件物品体积  $\geq 0$ 。取两种情况里面较大的作为  $dp[i][j]$  的值。

第二种方法  $dp[i][j]$  表示前  $i$  件物品能否填满容量为  $j$  的背包。分两种情况：一种情况为不要第  $i$  件物品，则  $dp[i][j] = dp[i-1][j]$ ；第二种情况为要第  $i$  件物品，则  $dp[i][j] = dp[i-1][j - A[i]]$ ，不过这种情况要求  $j - A[i] > 0$ 。取两种情况里面较大的作为  $dp[i][j]$  的值。

代码如下：

Solution 1:

```
public class Solution {
    /**
     * @param m: An integer m denotes the size of a backpack
     * @param A: Given n items with size A[i]
     * @return: The maximum size
     */
    public int backPack(int m, int[] A) {
        // write your code here
        if(m == 0 || A == null || A.length == 0){
            return 0;
        }

        int n = A.length;
        int[][] fillPack = new int[m + 1][n + 1];

        for(int i = 0; i <= m; i++){
            fillPack[i][0] = 0;
        }

        for(int j = 0; j <= n; j++){
            fillPack[0][j] = 0;
        }

        for(int i = 1; i <= m; i++){
            for(int j = 1; j <= n; j++){
                if(i - A[j - 1] >= 0){
                    //一种情况为不要第j件物品，则最优情况为背包容量为i时
                    //前j-1件物品的最优值；第二种情况为要第j件物品，则最优情况为背包容量为i-第j件
                    //物品体积时前j-1件物品的最优值加上第j件物品的值。
                    fillPack[i][j] = Math.max(fillPack[i][j - 1]
, fillPack[i - A[j - 1]][j - 1] + A[j - 1]);
                }else{
                    fillPack[i][j] = fillPack[i][j - 1];
                }
            }
        }
    }
}
```



```

        return fillPack[m][n];
    }
}

```

## Solution 2:

```

public class Solution {
    /**
     * @param m: An integer m denotes the size of a backpack
     * @param A: Given n items with size A[i]
     * @return: The maximum size
     */
    public int backPack(int m, int[] A) {
        boolean f[][] = new boolean[A.length + 1][m + 1];
        for (int i = 0; i <= A.length; i++) {
            for (int j = 0; j <= m; j++) {
                f[i][j] = false;
            }
        }
        f[0][0] = true;
        for (int i = 0; i < A.length; i++) {
            for (int j = 0; j <= m; j++) {
                f[i + 1][j] = f[i][j];
                if (j >= A[i] && f[i][j - A[i]]) {
                    f[i + 1][j] = true;
                }
            } // for j
        } // for i

        for (int i = m; i >= 0; i--) {
            if (f[A.length][i]) {
                return i;
            }
        }
        return 0;
    }
}

```



# Backpack II 125

## Question

Given  $n$  items with size  $A_i$  and value  $V_i$ , and a backpack with size  $m$ . What's the maximum value can you put into the backpack?

Notice

You cannot divide item into small pieces and the total size of items you choose should smaller or equal to  $m$ .

Example

Given 4 items with size  $[2, 3, 5, 7]$  and value  $[1, 5, 2, 4]$ , and a backpack with size 10. The maximum value is 9.

Challenge

$O(n \times m)$  memory is acceptable, can you do it in  $O(m)$  memory?

## Solution

和I类似。 $value[i][j]$ 表示容量为 $i$ 的背包在前 $j$ 件物品中能取的最大价值。分两种情况考虑：不要第 $j$ 件物品，则 $value[i][j] = value[i][j-1]$ ；要第 $j$ 件物品，则 $value[i][j] = value[i - A[j]][j-1] + V[j]$ ，这种情况要求 $i - A[j] \geq 0$ 。取两种情况里面较大的作为 $value[i][j]$ 的值。这种方法空间复杂度为 $O(n * m)$ 。

第二种方法可以将空间复杂度优化到 $O(m)$ 。 $f[j]$ 表示背包容量为 $j$ 时在前 $i$ 件物品中能够选取的最大价值。到第 $i$ 件物品时，考虑第 $i$ 件物品装还是不装。背包容量 $j$ 从最大的容量 $m$ 开始依次递减遍历，只考虑更新背包容量大于 $A[i]$ 的情况，即加上 $V[i]$ 之后是否结果会更优，背包容量小于 $A[i]$ 的情况不用考虑因为第 $i$ 件物品根本装不进此时容量的背包。若结果更优，则更新此时的 $f[j]$ （即装第 $i$ 件物品时情况更优），否则不变（即不装第 $i$ 件物品时情况更优）。因为 $f$ 会记录 $i$ 之前所有的最优情况，因此只需要 $m$ 空间即可。物品从0遍历到 $n-1$ ，即可求出前 $n$ 件物品的最优值。

其实第二种方法相当于使用滚动数组。因为状态函数为

$value[i][j] = \max(value[i - A[j]][j - 1] + V[j], value[i][j - 1])$

可以看出只和 $j-1$ 有关，因此保存两行信息就足够了。使用滚动数组必须逐行填写，因为在计算下一行的时候需要上一行的信息，像第一种方法中那样逐列填写就不行。

代码如下：

space  $O(n * m)$

```
public class Solution {
    /**
     * @param m: An integer m denotes the size of a backpack
     * @param A & V: Given n items with size A[i] and value V[i]
     * @return: The maximum value
     */
    public int backPackII(int m, int[] A, int V[]) {
        // write your code here
        if(m == 0 || A == null || A.length == 0){
            return 0;
        }

        int n = A.length;
        int[][] value = new int[m + 1][n + 1];

        for(int i = 1; i <= m; i++){
            for(int j = 1; j <= n; j++){
                if(i - A[j - 1] >= 0){
                    value[i][j] = Math.max(value[i][j - 1], value[i - A[j - 1]][j - 1] + V[j - 1]);
                }else{
                    value[i][j] = value[i][j - 1];
                }
            }
        }

        return value[m][n];
    }
}
```

space  $O(m)$  :

```
public class Solution {
    /**
     * @param m: An integer m denotes the size of a backpack
     * @param A & V: Given n items with size A[i] and value V[i]
     * @return: The maximum value
     */
    public int backPackII(int m, int[] A, int V[]) {
        // write your code here
        int[] f = new int[m + 1];

        for(int i = 0; i < n; i++){
            for(int j = m; j >= A[i]; j--){
                if(f[j] < f[j - A[i]] + V[i]){
                    f[j] = f[j - A[i]] + V[i];
                }
            }
        }

        return f[m];
    }
}
```

滚动数组：

```
public class Solution {
    /**
     * @param m: An integer m denotes the size of a backpack
     * @param A & V: Given n items with size A[i] and value V[i]
     * @return: The maximum value
     */
    public int backPackII(int m, int[] A, int V[]) {
        // write your code here
        if(m == 0 || A == null || A.length == 0){
            return 0;
        }

        int n = A.length;
        int[][] value = new int[m + 1][2];

        for(int j = 1; j <= n; j++){
            for(int i = 1; i <= m; i++){
                if(i - A[j - 1] >= 0){
                    value[i][j % 2] = Math.max(value[i][(j - 1)
% 2], value[i - A[j - 1]][(j - 1) % 2] + V[j - 1]);
                }else{
                    value[i][j % 2] = value[i][(j - 1) % 2];
                }
            }
        }
    }
}
```

# Backpack III 440

## Question

Given  $n$  kind of items with size  $A_i$  and value  $V_i$  (each item has an infinite number available) and a backpack with size  $m$ . What's the maximum value can you put into the backpack?

Notice

You cannot divide item into small pieces and the total size of items you choose should smaller or equal to  $m$ .

Example

Given 4 items with size  $[2, 3, 5, 7]$  and value  $[1, 5, 2, 4]$ , and a backpack with size 10. The maximum value is 15.

## Solution

这道题和II的思想一样， $f[j]$ 表示容量为 $j$ 的背包对前 $i$ 件物品能取的最大值，其中物品可以重复选取。对物品从0遍历到 $n-1$ ，每次只有比 $A[i]$ 大的背包容量才有可能被更新。

和II不同的是，这道题物品可以重复选择，所以内层遍历 $j$ 的时候从小到大遍历，这样物品可以重复选取。比如一开始在 $j$ 的时候取了 $i$ ，然后随着 $j$ 的增大，在 $j'$ 的时候又取了 $i$ ，而恰好 $j = j' - A[i]$ ，在这种情况下 $i$ 就被重复选取。如果从大往小遍历则所有物品只能取一次，所以II中是从大往小遍历。

因此可以重复取元素则背包容量从小到大遍历，反之从大到小遍历。

代码如下：

```
public class Solution {
    /**
     * @param A an integer array
     * @param V an integer array
     * @param m an integer
     * @return an array
     */
    public int backPackIII(int[] A, int[] V, int m) {
        // Write your code here
        int[] f = new int[m + 1];

        for(int i = 0; i < A.length; i++){
            for(int j = A[i]; j <= m; j++){
                //对于当前物品i，若j从小到大的话，很可能在j之前的j-A[i]时已经放过第i件物品了，在j时再放就是重复放入；若j从大到小，则j之前的所有情况都没有更新过，不可能放过第i件物品，所以不会重复放入。
                if(f[j - A[i]] + V[i] > f[j]){
                    f[j] = f[j - A[i]] + V[i];
                }
            }
        }

        return f[m];
    }
}
```



# Backpack IV 562

## Question

Given  $n$  items with size  $\text{nums}[i]$  which an integer array and all positive numbers, no duplicates. An integer target denotes the size of a backpack. Find the number of possible fill the backpack.

Each item may be chosen unlimited number of times.

Example

Given candidate items  $[2,3,6,7]$  and target 7,

A solution set is:

$[7]$

$[2, 2, 3]$

return 2

## Solution

这道题思路和III几乎一样， $\text{dp}[j]$ 表示背包容量为 $j$ 时，对前 $i$ 中物品来说能填满背包的方法数。当前元素为 $i$ 时，背包容量大于等于 $\text{nums}[i]$ 的才有可能被更新。此时，对于 $j$ 容量的背包，其新的方法数为前 $i-1$ 件物品能装满 $j$ 容量背包的方法数（即不装第 $i$ 件物品的方法数）+前 $i-1$ 件物品能装满 $j-\text{nums}[i]$ 容量的背包的方法数（即装第 $i$ 件物品的方法数）。这里状态方程只是把III中的 $\text{max}$ 改成了 $+$ 。所有求总共有多少种方法的题都可以从最大值问题变换 $\text{max}$ 为 $+$ 得到。因此，状态函数为：

$\text{dp}[j] = \text{dp}[j] + \text{dp}[j - \text{nums}[i]]$  (右边的 $\text{dp}[j]$ 表示上一行中（即 $i-1$ 件物品）能装满 $j$ 容量的方法数)

代码如下：

```
public class Solution {  
    /**  
     * @param nums an integer array and all positive numbers, no  
     * duplicates  
     * @param target an integer  
     * @return an integer  
     */  
    public int backPackIV(int[] nums, int target) {  
        // Write your code here  
        int[] dp = new int[target + 1];  
        dp[0] = 1;  
  
        for(int i = 0; i < nums.length; i++){  
            for(int j = nums[i]; j <= target; j++){  
                dp[j] += dp[j - nums[i]];  
            }  
        }  
  
        return dp[target];  
    }  
}
```

# Backpack V 563

## Question

Given  $n$  items with size  $\text{nums}[i]$  which an integer array and all positive numbers. An integer target denotes the size of a backpack. Find the number of possible fill the backpack.

Each item may only be used once.

Example

Given candidate items  $[1, 2, 3, 3, 7]$  and target 7,

A solution set is:

$[7]$

$[1, 3, 3]$

return 2

## Solution

这题和IV几乎一样, 不同的是元素只能取一次, 因此内层遍历 $j$ 的时候从大到小遍历 (解释见III)。  $\text{dp}[j]$ 表示背包容量为 $j$ 时, 对前 $i$ 件物品且每件物品只能取一次的情况下能取的最大值。  $\text{dp}[0]$ 解释一下: 就是将容量为0的背包装满的方法, 显然只有一种, 就是什么都不装。

代码如下:

```
public class Solution {  
    /**  
     * @param nums an integer array and all positive numbers  
     * @param target an integer  
     * @return an integer  
     */  
    public int backPackV(int[] nums, int target) {  
        // Write your code here  
        int[] dp = new int[target + 1];  
        dp[0] = 1;  
  
        for(int i = 0; i < nums.length; i++){  
            for(int j = target; j >= nums[i]; j--){  
                dp[j] += dp[j - nums[i]];  
            }  
        }  
  
        return dp[target];  
    }  
}
```

# Backpack VI 564

## Question

Given an integer array `nums` with all positive numbers and no duplicates, find the number of possible combinations that add up to a positive integer target.

Notice

The different sequences are counted as different combinations.

Example

Given `nums = [1, 2, 4]`, `target = 4`

The possible combination ways are:

[1, 1, 1, 1]

[1, 1, 2]

[1, 2, 1]

[2, 1, 1]

[2, 2]

[4]

return 6

## Solution

`dp[j]`表示背包容量为`j`时，数组`nums`能装满`j`的方法数量。

可以想到的方法是对于当前背包容量`j`，假设最后加入的元素为当前遍历的元素`i`，则将`i`元素最后加入的方法数量为`dp[j - nums[i]]`（此时要求`j - nums[i] >= 0`），将数组元素全部遍历当过`i`为止。其中，初始化`dp[0]=1`，表示数组元素将容量为0的背包装满的方法数为1（即什么元素都不取这一种方法）。

这道题并不难，但需要注意要和IV，V的区别。IV，V中的方法和元素位置有关系，元素的相对位置不能变化，不能先取后面的元素再取前面的元素，即相同元素组成的方法被视为一种方法（就算排列不同），其 $dp[j]$ 的含义为前 $i$ 件物品装满容量 $j$ 的方法数，因此只和 $i$ 之前的物品相关，而和 $i$ 之后的物品无关。这道题则说明不同的顺序被认为是不同的方法，因此和元素位置无关，可以先取后面的元素再取前面的元素，即相同元素的不同排列被视为不同的方法，其 $dp[j]$ 表示的是数组`nums`装满容量 $j$ 的方法数，是和数组中所有元素有关。

之前几题能够用一维 $dp$ 表示的本质其实是因为当前行的值只和上一行有关，因此用动态数组两行就行，如果直接在上一行更新当前行的状态则只需要一行即可，因此其本质就是还是二维 $dp$ 。但是这道题真的是一维 $dp$ ，即当前容量的填装数量只和之前容量填装的结果有关，只不过每次填装都要遍历整个`nums`数组来寻找相关的之前容量的状态，因此要用两重`for`循环。

代码如下：

```
public class Solution {
    /**
     * @param nums an integer array and all positive numbers, no
     * duplicates
     * @param target an integer
     * @return an integer
     */
    public int backPackVI(int[] nums, int target) {
        // Write your code here
        int[] dp = new int[target + 1];
        dp[0] = 1;

        for(int i = 1; i <= target; i++){
            for(int j = 0; j < nums.length; j++){
                if(i - nums[j] >= 0){
                    dp[i] += dp[i - nums[j]];
                }
            }
        }
        return dp[target];
    }
}
```



# House Robber 392

## Question

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.

Example

Given [3, 8, 4], return 8.

Challenge

$O(n)$  time and  $O(1)$  memory.

## Solution

$val[i]$ 表示抢第 $i$ 幢房子时能取得的最大值。有两种情况，即抢第 $i$ 幢房子，则其最大值为 $val[i-2]+A[i-1]$ ，或者不抢第 $i$ 幢房子，则其最大值为 $val[i-1]$ ，比较两者取大的即可。

状态函数： $val[i]=\max(val[i-1], val[i-2]+A[i-1])$

但是题目要求用 $O(1)$ 的space，所以需要用滚动数组优化，因为对于每个 $i$ ，只和 $i-1$ 以及 $i-2$ 相关，因此用长度为2的数组即可，满足题意。

代码如下：



```
public class Solution {
    /**
     * @param A: An array of non-negative integers.
     * return: The maximum amount of money you can rob tonight
     */
    public long houseRobber(int[] A) {
        // write your code here
        if(A == null || A.length == 0){
            return 0;
        }

        int n = A.length;
        long[] val = new long[2];
        val[0] = 0;
        val[1] = A[0];

        for(int i = 2; i <= n; i++){
            val[i % 2] = Math.max(val[(i - 1) % 2], val[(i - 2)
% 2] + A[i - 1]);
        }

        return val[n % 2];
    }
}
```

# House Robber II 534

## Question

After robbing those houses on that street, the thief has found himself a new place for his thievery so that he will not get too much attention. This time, all houses at this place are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, the security system for these houses remain the same as for those in the previous street.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.

Example

nums = [3,6,4], return 6

## Solution

这道题和I的思路一样，但是因为有环，所以不能同时抢第一家 and 最后一家。因此，只要把第一家或者最后一家去掉，再用I的方法求剩下的最大值，最后再在两种情况里面取大的那个即可。

代码如下：

```
public class Solution {  
    /**  
     * @param nums: An array of non-negative integers.  
     * return: The maximum amount of money you can rob tonight  
     */  
    public int houseRobber2(int[] nums) {  
        // write your code here  
        if(nums == null || nums.length == 0){  
            return 0;  
        }  
    }  
}
```

```
        int n = nums.length;
        if(n == 1){
            return nums[0];
        }

        int[] num1 = new int[n - 1];
        int[] num2 = new int[n - 1];
        for(int i = 0; i < n - 1; i++){
            num1[i] = nums[i];
            num2[i] = nums[i + 1];
        }

        return Math.max(houseRobber1(num1), houseRobber1(num2));
    }

    private int houseRobber1(int[] nums){
        if(nums == null || nums.length == 0){
            return 0;
        }

        int n = nums.length;
        int[] dp = new int[2];
        dp[0] = 0;
        dp[1] = nums[0];

        for(int i = 2; i <= n; i++){
            dp[i % 2] = Math.max(dp[(i - 1) % 2], dp[(i - 2) % 2]
] + nums[i - 1]);
        }

        return dp[n % 2];
    }
}
```

# House Robber III 535

## Question

The thief has found himself a new place for his thievery again. There is only one entrance to this area, called the "root." Besides the root, each house has one and only one parent house. After a tour, the smart thief realized that "all houses in this place forms a binary tree". It will automatically contact the police if two directly-linked houses were broken into on the same night.

Determine the maximum amount of money the thief can rob tonight without alerting the police.

Example

3 / \ 2 3 \ \ 3 1

Maximum amount of money the thief can rob = 3 + 3 + 1 = 7.

3

/ \ 4 5 / \ \ 1 3 1

Maximum amount of money the thief can rob = 4 + 5 = 9.

## Solution

这题和I，II不一样，不用dp，用递归。

每个节点都有两个选择，即选当前这个节点或者不选当前这个节点。如果选当前节点则不能选左右子节点，如果不选当前节点，则可以选左右子节点。所以对于每个节点，先递归求解其左右节点的值，若不选当前节点，则其最大值为 $\max(\text{选左节点, 不选左节点}) + \max(\text{选右节点, 不选右节点})$ ；若选当前节点，则其最大值为不选左节点 + 不选右节点 + 当前节点值。最后再在这两个里面取较大的那个即可。

代码如下：

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: The maximum amount of money you can rob tonight
     */
    public int houseRobber3(TreeNode root) {
        // write your code here
        int[] ans = dp(root);
        return Math.max(ans[0], ans[1]);
    }

    private int[] dp(TreeNode root){
        if(root == null){
            return new int[]{0, 0};
        }

        int[] left = dp(root.left);
        int[] right = dp(root.right);

        //0表示不选当前节点，1表示选当前节点
        int[] now = new int[2];
        now[0] = Math.max(left[0], left[1]) + Math.max(right[0],
right[1]);
        now[1] = left[0] + right[0] + root.val;
        return now;
    }
}
```

# Paint Fence 514

## Question

There is a fence with  $n$  posts, each post can be painted with one of the  $k$  colors. You have to paint all the posts such that no more than two adjacent fence posts have the same color. Return the total number of ways you can paint the fence.

Notice

$n$  and  $k$  are non-negative integers.

Example

Given  $n=3$ ,  $k=2$  return 6

```
post 1,    post 2, post 3
```

way1 0 0 1

way2 0 1 0

way3 0 1 1

way4 1 0 0

way5 1 0 1

way6 1 1 0

## Solution

用DP， $DP[i]$ 表示第 $i$ 个柱子最多的选择数。在计算 $DP[i]$ 时，考虑两种情况：

1. 和第 $i-1$ 柱子不同颜色，则可以有 $(k-1) * DP[i-1]$ 个选择

2. 和第 $i-1$ 柱子相同颜色，此时要求 $i-1$ 柱子和 $i-2$ 柱子不同颜色（即第一种情况，只是换成了第 $i-1$ 根柱子和第 $i-2$ 根柱子不同颜色），所以有 $(k-1) * DP[i-2]$ 个选择

3. 因此总选择数为 $(k-1) * (DP[i-1] + DP[i-2])$

因为只和前两个柱子相关，所以可以用滚动数组来优化空间。

代码如下：

```
public class Solution {
    /**
     * @param n non-negative integer, n posts
     * @param k non-negative integer, k colors
     * @return an integer, the total number of ways
     */
    public int numWays(int n, int k) {
        // Write your code here
        if(n > 2 && k == 1){
            return 0;
        }

        if(n == 1){
            return k;
        }

        int factor = k - 1;
        int[] dp = new int[3];
        dp[0] = k;
        dp[1] = k * k;

        for(int i = 2; i < n; i++){
            dp[i % 3] = factor * (dp[(i - 1) % 3] + dp[(i - 2) % 3]);
        }

        return dp[(n - 1) % 3];
    }
}
```





# Paint House 515

## Question

There are a row of  $n$  houses, each house can be painted with one of the three colors: red, blue or green. The cost of painting each house with a certain color is different. You have to paint all the houses such that no two adjacent houses have the same color.

The cost of painting each house with a certain color is represented by a  $n \times 3$  cost matrix. For example,  $\text{costs}[0][0]$  is the cost of painting house 0 with color red;  $\text{costs}[1][2]$  is the cost of painting house 1 with color green, and so on... Find the minimum cost to paint all houses.

Notice

All costs are positive integers.

Example

Given costs =  $[[14,2,11],[11,14,5],[14,3,10]]$  return 10

house 0 is blue, house 1 is green, house 2 is blue,  $2 + 5 + 3 = 10$

## Solution

这道题因为只有3中颜色，所以很简单。 $\text{dp}[i][j]$ 表示第 $i$ 幢房子涂 $j$ 的颜色最小的总和，即从前一幢房子的状态 $\text{dp}[i-1][k]$  ( $k \neq j$ )中选一个最小的再加上给第 $i$ 幢房子涂 $j$ 颜色的cost。如果直接在costs上修改，则不用单独开dp的空间，可以优化空间。

时间复杂度 $O(n)$ ，空间 $O(1)$ 。

代码如下：

```
public class Solution {
    /**
     * @param costs n x 3 cost matrix
     * @return an integer, the minimum cost to paint all houses
     */
    public int minCost(int[][] costs) {
        // Write your code here
        if(costs == null || costs.length == 0 || costs[0].length
== 0){
            return 0;
        }

        int n = costs.length;
        //直接 在原数组上修改不用耗费额外空间
        for(int i = 1; i < n; i++){
            costs[i][0] = costs[i][0] + Math.min(costs[i - 1][1]
, costs[i - 1][2]);
            costs[i][1] = costs[i][1] + Math.min(costs[i - 1][0]
, costs[i - 1][2]);
            costs[i][2] = costs[i][2] + Math.min(costs[i - 1][0]
, costs[i - 1][1]);
        }

        return Math.min(costs[n - 1][0], Math.min(costs[n - 1][1]
, costs[n - 1][2]));
    }
}
```

# Paint House II 516

## Question

There are a row of  $n$  houses, each house can be painted with one of the  $k$  colors. The cost of painting each house with a certain color is different. You have to paint all the houses such that no two adjacent houses have the same color.

The cost of painting each house with a certain color is represented by a  $n \times k$  cost matrix. For example,  $\text{costs}[0][0]$  is the cost of painting house 0 with color 0;  $\text{costs}[1][2]$  is the cost of painting house 1 with color 2, and so on... Find the minimum cost to paint all houses.

Notice

All costs are positive integers.

Example

Given  $n = 3$ ,  $k = 3$ ,  $\text{costs} = [[14,2,11],[11,14,5],[14,3,10]]$  return 10

house 0 is color 2, house 1 is color 3, house 2 is color 2,  $2 + 5 + 3 = 10$

Challenge

Could you solve it in  $O(nk)$ ?

## Solution

这道题思路和I十分相似，不同的是颜色变为 $k$ 种，若还是按照I中的方法，更新当前的某个值时需要搜索上一行和当前不同列的所有可能，则当 $k$ 很大时，会十分耗费时间。因此，可以不能直接用I中的方法，需要优化。

优化的方法为：

1. 只记录3个值，即前一行最小值，第二小值，和最小值的index。

2. 更新当前行元素，当前行元素记录的是当前行房子涂该种颜色时的最小值，只可能由该元素与上一行的最小值或次小值加和得到。遍历当前行的每个元素，若该元素的列和上一行最小值index不同，则更新为当前行元素值+上一行最小值，若和上一行最小值index相同，则更新为当前元素值+上一行次小值。同时将该值和当前行的最小值和次小值比较，更新当前行的最小值，次小值和最小值的index。
3. 重复2直到遍历所有行。

时间 $O(N=nk)$ ，空间 $O(1)$ 。

代码如下：

```
public class Solution {
    /**
     * @param costs n x k cost matrix
     * @return an integer, the minimum cost to paint all houses
     */
    public int minCostII(int[][] costs) {
        // Write your code here
        if(costs == null || costs.length == 0 || costs[0].length == 0){
            return 0;
        }

        int n = costs.length;
        int k = costs[0].length;

        int preLeast = 0;
        int preSecond = 0;
        int preIndex = -1;
        for(int i = 0; i < n; i++){
            int curLeast = Integer.MAX_VALUE;
            int curSecond = Integer.MAX_VALUE;
            int curIndex = -1;
            for(int j = 0; j < k; j++){
                if(j == preIndex){
                    costs[i][j] += preSecond;
                }else{
                    costs[i][j] += preLeast;
                }
            }
        }
    }
}
```

```
        }
        //更新最小值，次小值，最小值index
        if(costs[i][j] < curLeast){
            curSecond = curLeast;
            curLeast = costs[i][j];
            curIndex = j;
        }else if(costs[i][j] < curSecond){
            curSecond = costs[i][j];
        }
    }
    preLeast = curLeast;
    preSecond = curSecond;
    preIndex = curIndex;
}

return preLeast;
}
}
```

# Coins in a Line 394

## Question

There are  $n$  coins in a line. Two players take turns to take one or two coins from right side until there are no more coins left. The player who take the last coin wins.

Could you please decide the first play will win or lose?

Example

$n = 1$ , return true.

$n = 2$ , return true.

$n = 3$ , return false.

$n = 4$ , return true.

$n = 5$ , return true.

Challenge

$O(n)$  time and  $O(1)$  memory

## Solution

我的方法：谁走3谁就输。因此，如果 $n$ 能被3整除，则不管play1每次走几步，play2总能逼迫他走3，所以play1一定输，反之则play2一定输。结果是对的，但是没法证明。

DP：记忆化搜索。

假设在这一步我们取1个，则对手有两种选择：取1个或者2个。所以若 $n-2$ (我们1对手1)以及 $n-3$ (我们1对手2)都能获胜，则我们一定能获胜。

假设在这一步我们取2个，则对手有两种选择：取1个或者2个。所以若 $n-3$ (我们2对手1)以及 $n-4$ (我们2对手2)都能获胜，则我们一定能获胜。

只要上面两种情况有一种发生，则我们一定能获胜，反之则不能。

这里其实还可以考虑若 $n-1$ （我们取1个）和 $n-2$ （我们取2个）都不能获胜（即对手一定不能获胜），则我们一定获胜。但是会造成递归过深而爆栈。

代码如下：

time  $O(1)$  space  $O(1)$

```
public class Solution {
    /**
     * @param n: an integer
     * @return: a boolean which equals to true if the first player will win
     */
    public boolean firstWillWin(int n) {
        // write your code here
        if(n <= 0){
            return false;
        }

        if(n % 3 != 0){
            return true;
        }
        return false;
    }
}
```

DP

```
public class Solution {
    /**
     * @param n: an integer
     * @return: a boolean which equals to true if the first player will win
     */
    public boolean firstWillWin(int n) {
        // write your code here
        // if(n % 3 != 0){
        //     return true;
        // }
    }
}
```

```
        // return false;

        boolean[] dp = new boolean[n + 1];
        boolean[] visit = new boolean[n + 1];

        return search(n, dp, visit);
    }

    private boolean search(int n, boolean[] dp, boolean[] visit)
    {
        if(visit[n]){
            return dp[n];
        }

        if(n == 0){
            dp[n] = false;
        }else if (n == 1 || n == 2){
            dp[n] = true;
        }else{
            if((search(n - 2, dp, visit) && search(n - 3, dp, visit)) || (search(n - 3, dp, visit) && search(n - 4, dp, visit))){
                dp[n] = true;
            }else{
                dp[n] = false;
            }
        }

        visit[n] = true;
        return dp[n];
    }
}
```



# Coins in a Line II 395

## Question

There are  $n$  coins with different value in a line. Two players take turns to take one or two coins from left side until there are no more coins left. The player who take the coins with the most value wins.

Could you please decide the first player will win or lose?

Example

Given values array  $A = [1, 2, 2]$ , return true.

Given  $A = [1, 2, 4]$ , return false.

## Solution

1. 用DP解决。

$DP[i]$ 表示从 $i$ 到end能取到的最大值。

当走到 $i$ 时，我们有两种选择：

1. 取 $values[i]$ 的值，则对手可以取 $values[i+1]$ 的值或者 $values[i+1]+values[i+2]$ 的值：

1) 对手取 $values[i+1]$ 的值，我们在之后能取的最大值为 $DP[i+2]$

2) 对手取 $values[i+1]+values[i+2]$ 的值，我们在之后能取的最大值为 $DP[i+3]$

对手肯定希望最小化我们能取的值，所以我们能取的值是：

$values1 = values[i] + \min(DP[i+2], DP[i+3])$

1. 取 $values[i]+values[i+1]$ 的值，则对手可以取 $values[i+2]$ 的值或者 $values[i+2]+values[i+3]$ 的值：

1) 对手取 $values[i+2]$ 的值，我们在之后能取的最大值为 $DP[i+3]$

2) 对手取 $values[i+2]+values[i+3]$ 的值，我们在之后能取的最大值为 $DP[i+4]$

对手肯定希望最小化我们能取的值，所以我们能取的值是：

$values1 = values[i] + values[i+1] + \min(DP[i+3], DP[i+4])$

所以， $DP[i]$ 的值为 $\max(value1, value2)$ 。

因此我们可以初始化DP最后的几个值，然后倒着回来一直求道 $DP[0]$ 的值则为我们从0出发能取到的最大值。用所有数的总和减去 $DP[0]$ 即为对手能取到的值，比较两者大小，若 $DP[0]$ 更大则我们胜，否则对手胜。

### 1. 用memory search解。

$dp[i][j]$ 表示在 $i-j$ 区间里我们能取到的最大值。

我们每次能取值为1个或者2个，因此我们在 $i-j$ 区间能取的最大值为： $i-j$ 区间元素的总和—对手在剩下区间能取到的元素的总和的较小值。但是要注意当区间只有两个或以下元素时，直接取走剩下全部元素即可。因此，状态函数为：

$dp[i][j] = sum[i][j] - \min(dp[i+1][j], dp[i+2][j])$  当 $j-i \geq 2$ 时

$dp[i][j] = sum[i][j]$  当 $j-i < 2$ 时

注意一定要注意 $j-i$ 的范围，小心数组越界。

有一点疑惑：一开始用了一种方法，即将数组中从第3个数开始所有位置为3的倍数的数加和的值与剩下其它数加和的值进行比较，若前者大则输，反之则胜，也能得到正确结果。当时考虑的是对手一定能取到位置为3的倍数的数，若这些数之和比其它数之和大则对手一定能胜。

代码如下：

```
public class Solution {
    /**
     * @param values: an array of integers
     * @return: a boolean which equals to true if the first player will win
     */
    public boolean firstWillWin(int[] values) {
        // write your code here
        if(values == null || values.length == 0){
            return false;
        }
    }
}
```

```

        if(values.length < 3){
            return true;
        }

        int n = values.length;
        int[] DP = new int[n];
        DP[n - 1] = values[n - 1];
        DP[n - 2] = values[n - 2] + values[n - 1];
        DP[n - 3] = values[n - 3] + values[n - 2];

        for(int i = n - 4; i >= 0; i--){
            if(i + 4 >= n){
                DP[i] = Math.max(values[n - 4] + DP[n - 1], values[n - 4] + values[n - 3]);
                continue;
            }
            DP[i] = Math.max((values[i] + Math.min(DP[i + 2], DP[i + 3])), (values[i] + values[i + 1] + Math.min(DP[i + 3], DP[i + 4]))));
        }

        int sum = 0;
        for(int i = 0; i < n; i++){
            sum += values[i];
        }

        if(DP[0] > sum - DP[0]){
            return true;
        }
        return false;
    }
}

```

memory search

```

public class Solution {
    /**
     * @param values: an array of integers
     * @return: a boolean which equals to true if the first play

```

```

er will win
    */
    public boolean firstWillWin(int[] values) {
        // write your code here
        if(values == null || values.length == 0){
            return false;
        }

        if(values.length < 3){
            return true;
        }

        int n = values.length;
        int[][] dp = new int[n + 1][n + 1];
        boolean[][] visit = new boolean[n + 1][n + 1];

        int[] sum = new int[n + 1];
        sum[0] = 0;
        for(int i = 1; i <= n; i++){
            sum[i] = sum[i - 1] + values[i - 1];
        }

        return search(1, n, sum, dp, visit) > sum[n] / 2;
    }

    private int search(int start, int end, int[] sum, int[][] dp
, boolean[][] visit){
        if(visit[start][end]){
            return dp[start][end];
        }

        if(end - start >= 2){
            dp[start][end] = (sum[end] - sum[start - 1]) - Math.
min(search(start + 1, end, sum, dp, visit), search(start + 2, en
d, sum, dp, visit));
        }else{
            dp[start][end] = sum[end] - sum[start - 1];
        }

        visit[start][end] = true;
    }

```

```
        return dp[start][end];  
    }  
}
```

## Coins in a Line III 396

### Question

There are  $n$  coins in a line. Two players take turns to take a coin from one of the ends of the line until there are no more coins left. The player with the larger amount of money wins.

Could you please decide the first player will win or lose?

Example

Given array  $A = [3, 2, 2]$ , return true.

Given array  $A = [1, 2, 4]$ , return true.

Given array  $A = [1, 20, 4]$ , return false.

Challenge

Follow Up Question:

If  $n$  is even. Is there any hacky algorithm that can decide whether first player will win or lose in  $O(1)$  memory and  $O(n)$  time?

### Solution

用memory search的方法解，和II类似。 $dp[i][j]$ 表示在 $i-j$ 区间里我们能取到的最大值。

我们每次能取的值为开头或者结尾元素，因此我们在 $i-j$ 区间能取的最大值为： $i-j$ 区间元素的总和 - 对手在剩下区间能取到的元素的总和的较小值。因此，状态函数为：

$dp[i][j] = \text{sum}[i][j] - \min(dp[i+1][j], dp[i][j-1])$ 。

代码如下：

```
public class Solution {
```

```
/**
 * @param values: an array of integers
 * @return: a boolean which equals to true if the first player will win
 */
public boolean firstWillWin(int[] values) {
    // write your code here
    if(values == null || values.length == 0){
        return false;
    }

    int n = values.length;
    int[] sum = new int[n + 1];
    sum[0] = 0;
    for(int i = 1; i <= n; i++){
        sum[i] = sum[i - 1] + values[i - 1];
    }
    int[][] dp = new int[n + 1][n + 1];
    boolean[][] visit = new boolean[n + 1][n + 1];

    return search(1, n, sum, dp, visit) > sum[n] / 2;
}

private int search(int start, int end, int[] sum, int[][] dp, boolean[][] visit){
    if(visit[start][end]){
        return dp[start][end];
    }

    if(start == end){
        visit[start][end] = true;
        return dp[start][end] = sum[end] - sum[start - 1];
    }

    int max = (sum[end] - sum[start - 1]) - Math.min(search(start, end - 1, sum, dp, visit), search(start + 1, end, sum, dp, visit));

    visit[start][end] = true;
    dp[start][end] = max;
}
```

```
        return dp[start][end];  
    }  
}
```



# Longest Increasing Continuous Subsequence 397

## Question

Give an integer array , find the longest increasing continuous subsequence in this array.

An increasing continuous subsequence:

Can be from right to left or from left to right.

Indices of the integers in the subsequence should be continuous.

Notice

$O(n)$  time and  $O(1)$  extra space.

Example

For [5, 4, 2, 1, 3], the LICs is [5, 4, 2, 1], return 4.

For [5, 1, 2, 3, 4], the LICs is [1, 2, 3, 4], return 4.

## Solution

从前往后和从后往前各找一次，如果后一个（前一个）比前一个（后一个）大，则将length更新+1，否则将length更新为1（即从当前元素开始新的搜索）。

代码如下：

```
public class Solution {  
    /**  
     * @param A an array of Integer  
     * @return an integer  
     */  
    public int longestIncreasingContinuousSubsequence(int[] A) {  
        // Write your code here  
    }  
}
```

```
    if(A == null || A.length == 0){
        return 0;
    }

    if(A.length == 1){
        return 1;
    }

    int length = 1;
    int max = 0;
    for(int i = 1; i < A.length; i++){
        if(A[i] > A[i - 1]){
            length++;
        }else{
            length = 1;
        }
        max = Math.max(max, length);
    }

    length = 1;
    for(int i = A.length - 2; i >= 0; i--){
        if(A[i] > A[i + 1]){
            length++;
        }else{
            length = 1;
        }
        max = Math.max(max, length);
    }

    return max;
}
}
```

# Longest Increasing Continuous subsequence II 398

## Question

Give you an integer matrix (with row size  $n$ , column size  $m$ ) , find the longest increasing continuous subsequence in this matrix. (The definition of the longest increasing continuous subsequence here can start at any row or column and go up/down/right/left any direction).

Example

Given a matrix:

```
[ [1 ,2 ,3 ,4 ,5],  
  [16,17,24,23,6],  
  [15,18,25,22,7],  
  [14,19,20,21,8],  
  [13,12,11,10,9] ]
```

return 25

Challenge

$O(nm)$  time and memory.

## Solution

这道题用记忆化搜索，因为初始状态难以确定以及转移函数复杂，用dp非常困难。

用一个和A对应的二维数组L来记录以当前元素为终点的最长连续子序列的长度，如L[i][j]表示以L[i][j]为终点的最长子序列的长度。过程如下：

1. 遍历数组每个元素，寻找以当前元素为终点的最长连续子序列的长度。

2. 对当前元素做上下左右搜索，当其四周元素比当前元素小时，则对其四周元素继续搜索。
3. 在2中，若要搜索的元素之前已经被搜索过，则直接返回结果，否则继续搜索其四周元素。这样可以避免重复搜索。
4. 可以单独再开一个二维数组来记录每个元素是否被搜索过，这样比较费空间但省时间。也可以对L的每个元素初始化为负数，若搜索过则会大于等于0，因此可以间接记录元素是否曾经被搜索过，这样需要额外初始化的时间，但是省空间。

代码如下：

```
public class Solution {
    /**
     * @param A an integer matrix
     * @return an integer
     */
    int[] dx = {-1, 1, 0, 0};
    int[] dy = {0, 0, -1, 1};
    int n;
    int m;
    int[][] L;
    public int longestIncreasingContinuousSubsequenceII(int[][]
A) {
        // Write your code here
        if(A == null || A.length == 0 || A[0].length == 0){
            return 0;
        }

        int max = 0;
        n = A.length;
        m = A[0].length;
        L = new int[n][m];
        for(int i = 0; i < n; i++){
            Arrays.fill(L[i], -1);
        }

        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
```

```
        L[i][j] = search(A, i, j);
        if(L[i][j] > max){
            max = L[i][j];
        }
    }
}

return max;
}

private int search(int[][] A, int x, int y){
    if(L[x][y] >= 0){
        return L[x][y];
    }

    int res = 1;
    for(int i = 0; i < 4; i++){
        int curtx = x + dx[i];
        int curty = y + dy[i];
        if(curtx >= 0 && curtx < n && curty >= 0 && curty <
m && A[curtx][curty] < A[x][y]){
            int temp = search(A, curtx, curty) + 1;
            if(temp > res){
                res = temp;
            }
        }
    }
    L[x][y] = res;
    return res;
}
}
```

# Maximal Square

## Question

Given a 2D binary matrix filled with 0's and 1's, find the largest square containing all 1's and return its area.

Example

For example, given the following matrix:

```
1 0 1 0 0 1 0 1 1 1 1 1 1 1 1 0 0 1 0
```

Return 4.

## Solution

正方形可以一用一个角加边长来确定。遍历数组，以数组元素为正方形右下角，求出最大边长即可求出最大面积。

状态：`square[i][j]`代表以`matrix[i-1][j-1]`为右下角的正方形最长的边长。

转移函数：

```
1. square[i][j]=min(square[i-1][j], square[i-1][j-1], square[i][j-1])+1
   (matrix[i-1][j-1] == 1)
```

```
2. square[i][j]=0 (matrix[i-1][j-1] == 0)
```

每个正方形的左边，上面，左上正方形最大边长决定了该正方形的最大边长

初始化：额外多创建一行和一系列，并初始化为0

结果：遍历过程中每次取最大边长之积

**tips:** 因为每个正方形只和其左边，上面，左上正方形有关，因此可以用rolling array来进行空间优化，只用两行来记录即可，更新偶数行看奇数行，更新奇数行看偶数行。

代码如下：

```
public class Solution {
    /**
     * @param matrix: a matrix of 0 and 1
     * @return: an integer
     */
    public int maxSquare(int[][] matrix) {
        // write your code here
        if(matrix == null || matrix.length == 0 || matrix[0].length == 0){
            return 0;
        }
        int m = matrix.length;
        int n = matrix[0].length;
        //使用rolling array
        int[][] square = new int[2][n + 1];

        for(int i = 0; i < 2; i++){
            square[i][0] = 0;
        }

        for(int j = 0; j <= n; j++){
            square[0][j] = 0;
        }

        int max = 0;
        for(int i = 1; i <= m; i++){
            for(int j = 1; j <= n; j++){
                if(matrix[i - 1][j - 1] == 1){
                    square[i % 2][j] = Math.min(square[i % 2][j - 1], Math.min(square[(i - 1) % 2][j - 1], square[(i - 1) % 2][j])) + 1;
                }else{
                    square[i % 2][j] = 0;
                }
            }
        }
    }
}
```

```
                max = Math.max(max, square[i % 2][j] * square[i
% 2][j]);
            }
        }

        return max;
    }
}
```



# Maximal Rectangle 510

## Question

Given a 2D boolean matrix filled with False and True, find the largest rectangle containing all True and return its area.

Example

Given a matrix:

```
[[1, 1, 0, 0, 1],  
[0, 1, 0, 0, 1],  
[0, 0, 1, 1, 1],  
[0, 0, 1, 1, 1],  
[0, 0, 0, 0, 1]]
```

return 6.

## Solution

方法一：这道题是Largest Rectangle in Histogram直方图中最大的矩形问题的扩展，matrix有几行就有几个直方图。首先我们要构建“直方图”。

1) 第一行中，matrix值为0则相应直方图高度为0，matrix值为1则相应直方图高度为1

2) 从第二行开始，若matrix值为0则相应直方图高度为0，若matrix值为1则相应直方图高度为上一行直方图在该点的高度+1

直方图构建完成后，则只需要逐行调用Largest Rectangle in Histogram直方图中最大的矩形问题的函数并找出最大值即可。

将代码写在一个函数中可以更简洁。

方法二：这道题还能用DP来解。对于某个点 $(i, j)$ ，以该点为底边上的一点，以这个点向上能到达的高度的最大值为高，以在这条高度轴线上每一行向左右两边展开能到达的距离的最小值为宽的矩形为该点能取的矩形的最大值（好像并不是最大值，但一定会有一个点能取到最大的矩形）。用 $height[i][j]$ 来记录 $(i, j)$ 这个点向上能取的最大高度，用 $left[i][j]$ 来记录 $(i, j)$ 能取的最左边的左边界，用 $right[i][j]$ 来记录 $(i, j)$ 能取的最右边的右边界。因为当前点的各个值都只和上一行的点有关，因此要多建一行来初始化，同时可以用滚动数组来优化空间。

初始化：

```
height[0][j]=0  第一行所有点都为0
left[i][j]=0  假设每一个点的初始左边界都为0
right[i][j]=matrix[0].length - 1  假设每一个点的初始右边界都为matrix[0].length - 1
```

状态函数：从左至右扫描更新当前行的 $height$ 和 $left$

```
如果matrix[i-1][j]==1 (i-1是因为多建了一行便于计算)
height[i][j] = height[i-1][j]+1
left[i][j] = max(left[i-1][j], curLeft) curLeft为当前行的左边界，
每一行的左边界都是从0开始，直到遇到一个0，左边界就更新为j+1

如果matrix[i-1][j]==0
height[i][j] = 0
left[i][j] = 0
curLeft = j + 1 更新左边界
```

从右到左扫描更新当前行的 $right$

```
如果matrix[i-1][j]==1
right[i][j]=min(right[i-1][j], curRight) curRight为当前行的右边界，
每一行的右边界都是从matrix[0].length - 1开始，直到遇到一个0，右边界就更新为j-1

如果matrix[i-1][j]==0
right[i][j] = matrix[0].length - 1
curRight = j - 1 更新右边界
```

最后，再次扫描当前行每一个点，通过

```
height[i][j] * (right[i][j] - left[i][j] + 1)
```

来计算矩形面积，如果大于max，就更新。

可以用滚动数组来优化空间。

## Reference

代码如下：

```
public class Solution {
    /**
     * @param matrix a boolean 2D matrix
     * @return an integer
     */
    public int maximalRectangle(boolean[][] matrix) {
        // Write your code here
        if(matrix == null || matrix.length == 0 || matrix[0].length == 0){
            return 0;
        }

        int n = matrix.length;
        int m = matrix[0].length;
        int[][] height = new int[n][m];

        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                if(i == 0){
                    height[i][j] = (matrix[i][j] == false)? 0 : 1;
                }else{
                    height[i][j] = (matrix[i][j] == false)? 0 : (height[i - 1][j] + 1);
                }
            }
        }
    }
}
```

```

        int max = 0;
        for(int i = 0; i < n; i++){
            max = Math.max(max, largestRectangleArea(height[i]))
        }

        return max;
    }

    private int largestRectangleArea(int[] heights){
        Stack<Integer> stack = new Stack<Integer>();
        int area = 0;
        for(int i = 0; i < heights.length; i++){
            while(!stack.isEmpty() && heights[i] < heights[stack
                .peek()]){
                int pos = stack.pop();
                int w = stack.isEmpty()? i : i - stack.peek() -
1;

                int h = heights[pos];
                area = Math.max(area, w * h);
            }
            stack.push(i);
        }

        while(!stack.isEmpty()){
            int pos = stack.pop();
            int w = stack.isEmpty()? heights.length : heights.le
ngth - stack.peek() - 1;
            int h = heights[pos];
            area = Math.max(area, w * h);
        }

        return area;
    }
}

```

Revised Version:

```
public class Solution {
```

```
public int maximalRectangle(char[][] matrix) {
    if (matrix.length < 1) return 0;
    int n = matrix.length;
    if (n == 0) return 0;
    int m = matrix[0].length;
    if (m == 0) return 0;
    int[][] height = new int[n][m];

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (i == 0)
                height[i][j] = ((matrix[i][j] == '1') ? 1 :
0);
            else
                height[i][j] += ((matrix[i][j] == '1') ? hei
ght[i-1][j] + 1 : 0);
        }
    }

    int answer = 0;
    for (int i = 0; i < n; ++i) {
        Stack<Integer> S = new Stack<Integer>();
        for (int j = 0; j < m; j++) {
            while (!S.empty() && height[i][j] < height[i][S.
peek()]) {
                int pos = S.peek();
                S.pop();
                answer = Math.max(answer, height[i][pos]*(S.
empty()? j : j-S.peek()-1));
            }
            S.push(j);
        }
        while (!S.empty()) {
            int pos = S.peek();
            S.pop();
            answer = Math.max(answer, height[i][pos]*(S.empty
y()? m : m-S.peek()-1));
        }
    }
    return answer;
}
```

```

    }
}

```

DP:

```

public int maximalRectangle(boolean[][] matrix) {
    //Write your code here
    if(matrix == null || matrix.length == 0 || matrix[0].length == 0){
        return 0;
    }

    int n = matrix.length;
    int m = matrix[0].length;
    int[] height = new int[m];
    int[] left = new int[m];
    int[] right = new int[m];
    Arrays.fill(right, m - 1);

    int max = 0;
    for(int i = 0; i < n; i++){
        int curLeft = 0;
        int curRight = m - 1;

        for (int j = 0; j < m; j++) {
            if (matrix[i][j] == true) {
                height[j]++;
                left[j] = Math.max(left[j], curLeft);
            } else {
                height[j] = 0;
                left[j] = 0;
                curLeft = j + 1;
            }
        }

        for(int j = m - 1; j >= 0; j--){
            if(matrix[i][j] == true){
                right[j] = Math.min(right[j], curRight);
            }else{

```

```
        right[j] = m;
        curRight = j - 1;
    }
}

for(int j = 0; j < m; j++){
    int area = (right[j] - left[j] + 1) * height[j];
    max = Math.max(max, area);
}

return max;
}
```

# Unique Binary Search Trees 163

## Question

Given  $n$ , how many structurally unique BSTs (binary search trees) that store values  $1 \dots n$ ?

Example

Given  $n = 3$ , there are a total of 5 unique BST's.

1 3 3 2 1 \ / / / \ \ 3 2 1 1 3 2 / / \ \ 2 1 2 3

## Solution

思路很简单，就是以某个点 $i$ 作为根节点时，BST的数目为 $i$ 左边所有点的BST的个数 \*  $i$ 右边所有点的BST的个数

定义 $\text{Count}[i]$ 为 $i$ 个数能产生的Unique Binary Tree的数目，

如果数组为空，毫无疑问，只有一种BST，即空树，

```
Count[0] = 1
```

如果数组仅有一个元素 $\{1\}$ ，只有一种BST，单个节点

```
Count[1] = 1
```

如果数组有两个元素 $\{1,2\}$ ，那么有如下两种可能

```
Count[2] = Count[0] * Count[1]    (1为根，左边0个数，右边1个数)
          + Count[1] * Count[0]    (2为根，左边1个数，右边0个数)
```

再看一遍三个元素的数组，可以发现BST的取值方式如下：



```
Count[3] = Count[0]*Count[2]  (1为根，左边0个数，右边2个数)
          + Count[1]*Count[1]  (2为根，左边1个数，右边1个数)
          + Count[2]*Count[0]  (3为根，左边2个数，右边0个数)
```

所以，由此观察，可以得出Count的递推公式为

$$\text{Count}[n+1] = \sum \text{Count}[i] * [n-i]$$

问题至此划归为一维动态规划。

代码如下：

```
public class Solution {
    /**
     * @paramn n: An integer
     * @return: An integer
     */
    public int numTrees(int n) {
        // write your code here
        if(n <= 1){
            return 1;
        }

        int[] dp = new int[n + 1];
        dp[0] = 1;
        dp[1] = 1;

        for(int i = 2; i <= n; i++){
            for(int j = 0; j < i; j++){
                //Count[n+1] = ∑ Count[i] * [n-i]
                dp[i] += dp[j] * dp[i - j - 1];
            }
        }

        return dp[n];
    }
}
```



# Unique Binary Search Trees II 164

## Question

Given  $n$ , generate all structurally unique BST's (binary search trees) that store values  $1 \dots n$ .

Example

Given  $n = 3$ , your program should return all 5 unique BST's shown below.

1 3 3 2 1 \ / / / \ \ 3 2 1 1 3 2 / / \ \ 2 1 2 3

## Solution

用递归解，没用到DP？

依次遍历  $1-n$ ，以当前的数为根节点。递归寻找当前数左边的所有数形成的BST的根节点，以及当前的数右边的所有数形成的BST的根节点，然后再遍历左右的根节点，将其依次插入当前数形成的根节点的左右两边（设左边有  $m$  个根节点，右边有  $n$  个根节点，则一共有  $m \times n$  种插入方法，因此以当前数为根节点会形成  $m \times n$  种BST），然后将当前数根节点插入result。

代码如下：

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
```

```
/**
 * @paramn n: An integer
 * @return: A list of root
 */
public List<TreeNode> generateTrees(int n) {
    // write your code here
    return generate(1, n);
}

private List<TreeNode> generate(int start, int end){
    List<TreeNode> res = new ArrayList<TreeNode>();
    if(start > end){
        //将null加入res中，这样res永远不会为null，在返回的那层也就不需要讨论res为null的情况
        res.add(null);
        return res;
    }

    if(start == end){
        res.add(new TreeNode(start));
        return res;
    }

    for(int i = start; i <= end; i++){
        List<TreeNode> left = generate(start, i - 1);
        List<TreeNode> right = generate(i + 1, end);

        for(TreeNode leftRoot : left){
            for(TreeNode rightRoot : right){
                TreeNode root = new TreeNode(i);
                root.left = leftRoot;
                root.right = rightRoot;
                res.add(root);
            }
        }
    }

    return res;
}
```



# Unique Paths 114

## Question

A robot is located at the top-left corner of a  $m \times n$  grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?

Notice

$m$  and  $n$  will be at most 100.

## Solution

这题和minimum path sum一样要求从左上走到右下，只是这次要求的是一共有多少条路径。和climbing stairs的思想一样。

$\text{uniquePath}[i][j]$ 表示从  $(0, 0)$  到  $(i, j)$  的路径数之和。

初始化：

```
uniquePath[0][0] = 1,  
uniquePath[i][0]=uniquePath[i-1][0],  
uniquePath[0][j]=uniquePath[0][j-1]
```

状态函数：

```
uniquePath[i][j]=uniquePath[i-1][j]+uniquePath[i][j-1]
```

即从  $(0, 0)$  到达  $(i, j)$  的路径数为从  $(0, 0)$  到达其左边和上面的点的路径数之和。

代码如下：

```
public class Solution {
    /**
     * @param n, m: positive integer (1 <= n ,m <= 100)
     * @return an integer
     */
    public int uniquePaths(int m, int n) {
        // write your code here
        if(m <= 0 || n <= 0){
            return 0;
        }

        int[][] uniquePath = new int[m][n];
        uniquePath[0][0] = 1;

        for(int i = 1; i < m; i++){
            uniquePath[i][0] = uniquePath[i - 1][0];
        }

        for(int j = 1; j < n; j++){
            uniquePath[0][j] = uniquePath[0][j - 1];
        }

        for(int i = 1; i < m; i++){
            for(int j = 1; j < n; j++){
                uniquePath[i][j] = uniquePath[i][j - 1] + unique
Path[i - 1][j];
            }
        }

        return uniquePath[m - 1][n - 1];
    }
}
```

# Unique Paths II 115

## Question

Follow up for "Unique Paths":

Now consider if some obstacles are added to the grids. How many unique paths would there be?

An obstacle and empty space is marked as 1 and 0 respectively in the grid.

Notice

m and n will be at most 100.

Example

For example,

There is one obstacle in the middle of a 3x3 grid as illustrated below.

[ [0,0,0], [0,1,0], [0,0,0] ]

The total number of unique paths is 2.

## Solution

做法和Unique Paths I一样，不同点在于当遇到障碍物的时候直接将该点路径数设为0。

初始化：

```
uniquePath[0][0] = 1,  
当(i,0)不为障碍物时  
uniquePath[i][0]=uniquePath[i-1][0], 否则等于0  
当(0,j)不为障碍物时  
uniquePath[0][j]=uniquePath[0][j-1], 否则等于0
```



状态函数：

```
uniquePath[i][j]=uniquePath[i-1][j]+uniquePath[i][j-1] 当 (i,j)
不为障碍物时
uniquePath[i][j]=0 当 (i,j) 为障碍物时
```

代码如下：

```
public class Solution {
    /**
     * @param obstacleGrid: A list of lists of integers
     * @return: An integer
     */
    public int uniquePathsWithObstacles(int[][] obstacleGrid) {
        // write your code here
        if(obstacleGrid == null || obstacleGrid.length == 0){
            return 0;
        }

        if(obstacleGrid[0] == null || obstacleGrid[0].length ==
0){
            return 0;
        }

        if(obstacleGrid[0][0] == 1){
            return 0;
        }

        int m = obstacleGrid.length;
        int n = obstacleGrid[0].length;
        int[][] uniquePath2 = new int[m][n];
        uniquePath2[0][0] = 1;

        for(int i = 1; i < m; i++){
            if(obstacleGrid[i][0] == 0){
                uniquePath2[i][0] = 1;
            }else{
                break;
            }
        }
    }
}
```

```
    }

    for(int j = 1; j < n; j++){
        if(obstacleGrid[0][j] == 0){
            uniquePath2[0][j] = 1;
        }else{
            break;
        }
    }

    for(int i = 1; i < m; i++){
        for(int j = 1; j < n; j++){
            if(obstacleGrid[i][j] != 1){
                uniquePath2[i][j] = uniquePath2[i][j - 1] +
uniquePath2[i - 1][j];
            }else{
                uniquePath2[i][j] = 0;
            }
        }
    }

    return uniquePath2[m - 1][n - 1];
}
}
```

# Build Post Office 573

## Question

Given a 2D grid, each cell is either an house 1 or empty 0 (the number zero, one), find the place to build a post office, the distance that post office to all the house sum is smallest. Return the smallest distance. Return -1 if it is not possible.

Notice

You can pass through house and empty.

You only build post office on an empty.

Example

Given a grid:

```
0 1 0 0 1 0 1 1 0 1 0 0
```

return 6. (Placing a post office at (1,1), the distance that post office to all the house sum is smallest.)

## Solution

这道题可以用dp来做，但是会超时。首先扫描一遍将所有house的点记录下来，然后遍历图中所有0的点，计算每个0的点到这些house的距离和，选最小的那个即可。这种情况下可以优化到 $O(k * n^2)$ ，但是如果数据量很大还是过不了。

因此需要减少搜索的点。想到的方法是在所有房子围成的形状的重心位置附近建邮局则到所有房子的距离之和最短（怎么证明？）。因此步骤如下：

1. 首先找到所有房子的重心。找所有房子x值的median和y值的median（如果是奇数个就是排序后取中间值，如果是偶数则取中间两个数再取平均值）即为重心。

2. 然后用bfs来搜索。将重心加入queue中，然后开始一圈一圈（将出队的每个点周围八个点加入队中）向外找，用的是和逐层遍历二叉树的类似的方法（即在每一层开始的时候记录一下本层的点的个数，然后一次出队这么多点即可将本层的点全部出队）。每一圈结束时，返回该圈上的点作为post office能取的最小值，如果存在则停止搜索。即如果存在可以作为post office的点，则外圈点到各个房子的距离一定不会比内圈点更优。

代码如下：

median version

```
public class Solution {
    /**
     * @param grid a 2D grid
     * @return an integer
     */
    class Node{
        int x;
        int y;
        public Node(int x, int y){
            this.x = x;
            this.y = y;
        }
    }
    int[] dx = {0, 0, -1, 1, -1, 1, -1, 1};
    int[] dy = {-1, 1, 0, 0, -1, -1, 1, 1};

    //Median version
    public int shortestDistance(int[][] grid) {
        // Write your code here
        if(grid == null || grid.length == 0 || grid[0].length == 0){
            return -1;
        }

        int n = grid.length;
        int m = grid[0].length;
        boolean[][] visit = new boolean[n][m];
        ArrayList<Node> house = new ArrayList<Node>();
        ArrayList<Integer> xArr = new ArrayList<Integer>();
```

```
ArrayList<Integer> yArr = new ArrayList<Integer>();
//find house position
for(int i = 0; i < n; i++){
    for(int j = 0; j < m; j++){
        if(grid[i][j] == 1){
            house.add(new Node(i, j));
            xArr.add(i);
            yArr.add(j);
        }
    }
}
//no empty place
if(house.size() == m * n){
    return -1;
}

if(house.size() == 0){
    return 0;
}

//find the median of house positions
int xMedian = getMedian(xArr);
int yMedian = getMedian(yArr);

Queue<Node> queue = new LinkedList<Node>();
queue.add(new Node(xMedian, yMedian));
visit[xMedian][yMedian] = true;
int min = Integer.MAX_VALUE;
while(!queue.isEmpty()){
    int size = queue.size();
    for(int i = 0; i < size; i++){
        Node curt = queue.poll();
        if(grid[curt.x][curt.y] == 0){
            min = Math.min(min, search(house, curt));
        }
        for(int j = 0; j < 8; j++){
            int nextX = curt.x + dx[j];
            int nextY = curt.y + dy[j];
            if(nextX >= 0 && nextX < n && nextY >= 0 &&
nextY < m && !visit[nextX][nextY]){
```

```

        visit[nextX][nextY] = true;
        queue.add(new Node(nextX, nextY));
    }
}
}
if(min != Integer.MAX_VALUE){
    return min;
}
}

return -1;
}

private int getMedian(ArrayList<Integer> arr){
    Collections.sort(arr);

    int Median = arr.get(arr.size() / 2);

    if(arr.size() % 2 == 0){
        Median = (Median + arr.get(arr.size() / 2 - 1)) / 2;
    }

    return Median;
}

private int search(ArrayList<Node> house, Node curt){
    int sum = 0;
    for(Node node : house){
        sum += Math.abs(curt.x - node.x) + Math.abs(curt.y -
node.y);
    }
    return sum;
}
}

```

dp version

```

//DP version Time Limit Exceed
// public int shortestDistance(int[][] grid) {

```

```

        //      // Write your code here
        //      if(grid == null || grid.length == 0 || grid[0].length
== 0){
        //          return -1;
        //      }

        //      int n = grid.length;
        //      int m = grid[0].length;
        //      ArrayList<Node> house = new ArrayList<Node>();
        //      ArrayList<Node> empty = new ArrayList<Node>();
        //      //find house position
        //      for(int i = 0; i < n; i++){
        //          for(int j = 0; j < m; j++){
        //              if(grid[i][j] == 1){
        //                  house.add(new Node(i, j));
        //              }else{
        //                  empty.add(new Node(i, j));
        //              }
        //          }
        //      }
        //      //no empty place
        //      if(empty.size() == 0){
        //          return -1;
        //      }

        //      int min = Integer.MAX_VALUE;
        //      for(Node node : empty){
        //          min = Math.min(min, getDistance(node, house));
        //      }
        //      return min;
        //  }

        // private int getDistance(Node curt, ArrayList<Node> house)
{
        //      int sum = 0;
        //      for(Node node : house){
        //          sum += Math.abs(curt.x - node.x) + Math.abs(curt.
y - node.y);
        //      }
        //      return sum;

```

```
// }
```



# Build Post Office II 574

## Question

Given a 2D grid, each cell is either a wall 2, an house 1 or empty 0 (the number zero, one, two), find the place to build a post office, the distance that post office to all the house sum is smallest. Return the smallest distance. Return -1 if it is not possible.

Notice

You cannot pass through wall and house, but can pass through empty.

You only build post office on an empty.

Example

Given a grid:

```
0 1 0 0 0 1 0 0 2 1 0 1 0 0 0
```

return 8, You can build at (1,1). (Placing a post office at (1,1), the distance that post office to all the house sum is smallest.)

Challenge

Solve this problem within  $O(n^3)$  time.

## Solution

这道题和I比较类似，但是因为不能穿过wall和house，所以必须用bfs的方法搜索最近距离，而不能直接计算几何距离。

1. 将数组扫描一遍找到所有房子。
2. 为每一个房子建立一个距离矩阵，计算该房子到所有0点的距离。即 $distance[i][j][k]$ 为k房子到 $grid[i][j]$ 上的点的距离。计算距离的时候用bfs搜索。

3. 然后遍历图上所有为0的点，查询k张距离矩阵，将所有房子到该点的距离加起来即为在该点建邮局的距离总和。若在查询过程中遇到某个点在某张距离矩阵上的值为无穷大，则说明该点无法到达该房子，直接停止搜索即可。
4. 选3中距离最小的点即可。

代码如下：

```
public class Solution {
    /**
     * @param grid a 2D grid
     * @return an integer
     */
    class Node{
        int x;
        int y;
        int dis;
        public Node(int x, int y, int dis){
            this.x = x;
            this.y = y;
            this.dis = dis;
        }
    }

    public int shortestDistance(int[][] grid) {
        // Write your code here
        if(grid == null || grid.length == 0 || grid[0].length == 0){
            return -1;
        }

        int n = grid.length;
        int m = grid[0].length;
        ArrayList<Node> house = new ArrayList<Node>();
        //find house position
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                if(grid[i][j] == 1){
                    house.add(new Node(i, j, 0));
                }
            }
        }
    }
}
```

```

        }
    }
    //no empty place
    int k = house.size();
    if(k == n * m){
        return -1;
    }

    int[][][] distance = new int[k][n][m];
    for(int i = 0; i < k; i++){
        for(int j = 0; j < n; j++){
            Arrays.fill(distance[i][j], Integer.MAX_VALUE);
        }
    }

    for(int i = 0; i < k; i++){
        getDistance(house.get(i), distance, i, grid);
    }

    int min = Integer.MAX_VALUE;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            if(grid[i][j] == 0){
                int sum = 0;
                for(int l = 0; l < k; l++){
                    if(distance[l][i][j] == Integer.MAX_VALU
E){
                        sum = Integer.MAX_VALUE;
                        break;
                    }
                    sum += distance[l][i][j];
                }
                min = Math.min(min, sum);
            }
        }
    }

    if(min == Integer.MAX_VALUE){
        return -1;
    }

```

```

        return min;
    }

    int[] dx = {-1, 1, 0, 0};
    int[] dy = {0, 0, -1, 1};
    //BFS search for shortest path
    private void getDistance(Node curt, int[][][] distance, int
k, int[][] grid){
        int n = grid.length;
        int m = grid[0].length;
        Queue<Node> queue = new LinkedList<Node>();
        boolean[][] visited = new boolean[n][m];
        queue.offer(curt);
        visited[curt.x][curt.y] = true;

        while(!queue.isEmpty()){
            Node now = queue.poll();
            for(int i = 0; i < 4; i++){
                int nextX = now.x + dx[i];
                int nextY = now.y + dy[i];
                if(nextX >= 0 && nextX < n && nextY >= 0 && next
Y < m && grid[nextX][nextY] == 0 && !visited[nextX][nextY]){
                    distance[k][nextX][nextY] = now.dis + 1;
                    queue.add(new Node(nextX, nextY, now.dis + 1
));
                    visited[nextX][nextY] = true;
                }
            }
        }
    }
}

```

# Post Office Problem 435

## Question

On one line there are  $n$  houses. Give you an array of integer means the position of each house. Now you need to pick  $k$  position to build  $k$  post office, so that the sum distance of each house to the nearest post office is the smallest. Return the least possible sum of all distances between each village and its nearest post office.

Example

Given array  $a = [1,2,3,4,5]$ ,  $k = 2$ .

return 3.

Challenge

Could you solve this problem in  $O(n^2)$  time ?

## Solution

状态函数：

$$dp[i][l] = dp[j][l-1] + dis[j+1][i] \quad (l-1 \leq j < i)。$$

其中  $dp[i][l]$  表示在前  $i$  个村庄中建  $l$  个 post 的最短距离， $j$  为分隔点，可以将问题转化为在前  $j$  个村庄建  $l-1$  个 post 的最短距离 + 在第  $j+1$  到第  $i$  个村庄建 1 个 post 的最短距离。其中有个性性质，如元素是单调排列的，则在中间位置到各个元素的距离和最小。

1. 初始化  $dis$  矩阵，枚举不同开头和结尾的村庄之间建 1 个 post 的最小距离，即求出开头和结尾村庄的中间点，然后计算开头到结尾的所有点到中间点的距离。记得要对原矩阵排序，这样才能用中间点距离最小性质。
2. 初始化  $dp$  矩阵，即初始化  $dp[i][1]$ ，求前  $i$  个村庄建 1 个 post 的最小距离（可根据  $dis$  求出）。

3. post数 $l$ 从2枚举到 $k$ ，开始村庄 $i$ 从 $l$ 枚举到结尾（因为要建 $l$ 个post至少需要 $l$ 个村庄，否则没有意义），然后根据状态函数求 $dp[i][l]$ ，分割点 $j$ 从 $l-1$ 枚举到 $i-1$ （前 $j$ 个村庄建 $l-1$ 个post则至少需要 $l-1$ 个村庄），在这些分隔点的情况下求 $dp[i][l]$ 的最小值。

4. 返回 $dp[n][k]$ 即可。

代码如下：

dp  $O(n^3)$

```
public class Solution {
    /**
     * @param A an integer array
     * @param k an integer
     * @return an integer
     */
    private int[][] initial(int[] A){
        int n = A.length;
        int[][] dis = new int[n + 1][n + 1];
        for(int i = 1; i <= n; i++){
            for(int j = i + 1; j <= n; j++){
                int mid = (i + j) / 2;
                for(int k = i; k <= j; k++){
                    //所有点到mid的距离
                    dis[i][j] += Math.abs(A[k - 1] - A[mid - 1])
                }
            }
        }
        return dis;
    }

    public int postOffice(int[] A, int k) {
        // Write your code here
        if(A == null || A.length == 0 || k <= 0 || k >= A.length){
            return 0;
        }
    }
}
```

```

//一个Array只有单调才满足中间的数和其它所有数的差的绝对值之和最小
Arrays.sort(A);

int n = A.length;

//dis[i][j]: the distance sum if build post in the mid of the i,j
int[][] dis = initial(A);

//dp[i][l]: 前i个house建l个post的最小距离
int[][] dp = new int[n + 1][k + 1];

//只建一个post情况，在中间建距离之和最小
for(int i = 0; i <= n; i++){
    dp[i][1] = dis[1][i];
}

for(int l = 2; l <= k; l++){
    for(int i = l; i <= n; i++){
        dp[i][l] = Integer.MAX_VALUE;
        //j为分割点，从l-1到i-1(因为l-1个post至少需要l-1个house)
        for(int j = l - 1; j < i; j++){
            dp[i][l] = Math.min(dp[i][l], dp[j][l - 1] +
dis[j + 1][i]);
        }
    }
}

return dp[n][k];
}
}

```

# Perfect Squares 513

## Question

Given a positive integer  $n$ , find the least number of perfect square numbers (for example, 1, 4, 9, 16, ...) which sum to  $n$ .

Example

Given  $n = 12$ , return 3 because  $12 = 4 + 4 + 4$

Given  $n = 13$ , return 2 because  $13 = 4 + 9$

## Solution

这道题不难，但是想说清楚好难。。。

$dp[i]$ 表示 $i$ 可以由多少个完全平方数加和得到的最小值。

1. 将小于 $n$ 的完全平方数初始化为1，表示完全平方数可以由自己加和得到。
2. 因为所有的数都可以看成某个数加上一个完全平方数得到。因此 $i$ 从0遍历到 $n$ ，寻找 $i$ 加上某个完全平方数的和小于等于 $n$ 时的最小值。设当前数 $(i + j \cdot j)$ 可以由 $i$  + 某个完全平方数 $a = j \cdot j (j \cdot j \leq n)$ 得到，若 $dp[i] + dp[j \cdot j]$ （即得到 $i$ 所需要的最小值 + 得到 $j \cdot j$ 所需要的最小值）小于 $dp[i + j \cdot j]$ （因为 $j$ 也可以由另外一个数 $i'$ 加上另外一个完全平方数 $a'$ 得到），则更新 $dp[j \cdot j]$ 的值。

代码如下：



```
public class Solution {
    /**
     * @param n a positive integer
     * @return an integer
     */
    public int numSquares(int n) {
        // Write your code here
        int[] dp = new int[n + 1];
        Arrays.fill(dp, Integer.MAX_VALUE);
        for(int i = 0; i * i <= n; i++){
            dp[i * i] = 1;
        }

        for(int i = 0; i <= n; i++){
            for(int j = 0; i + j * j <= n; j++){
                dp[i + j * j] = Math.min(dp[i] + 1, dp[i + j * j
]);
            }
        }

        return dp[n];
    }
}
```

# Longest Palindromic Substring 200

## Question

Given a string S, find the longest palindromic substring in S. You may assume that the maximum length of S is 1000, and there exists one unique longest palindromic substring.

Example

Given the string = "abcdzdcab", return "cdzdc".

Challenge

$O(n^2)$  time is acceptable. Can you do it in  $O(n)$  time.

## Solution

三种方法。

第一种brute force，遍历每一个点i，同时遍历0-len的长度，并判断i-i+l是否为palindromic。时间复杂度 $O(n^3)$ ，完全不能接受。

第二种DP，思想为求i-j的substring是否为palindromic，只要判断i和j是否相同，同时i+1~j-1是否为palindromic。因此右边界j每次向右移一位，左边界i每次从0-j。时间复杂度为 $O(n^2)$ 。

第三种为中心展开，在字符串开头结尾和之间加上“#”，然后遍历新字符串，求以当前字符为中心的最长palindromic字符串的长度。方法为从当前字符串出发，向左右两边延伸，每次延伸判断延伸的两端新字符是否相等，一直延伸到不能再延伸为止。记录最长的字符串即为答案。该方法时间复杂度依然是 $O(n^2)$ ？

第三种方法的优化：<http://blog.csdn.net/hopeztm/article/details/7932245>。但是实现不出来。。。。

代码如下：

```
public class Solution {
```

```

/**
 * @param s input string
 * @return the longest palindromic substring
 */
//O(n^3) BF
// public String longestPalindrome(String s) {
//     // Write your code here
//     if(s == null || s.length() == 0){
//         return "";
//     }

//     int len = s.length();
//     int max = 0;
//     int[] index = new int[2];
//     for(int i = 0; i < len; i++){
//         for(int l = 1; l <= len; l++){
//             if(i + l <= len){
//                 if(l > max && isPalindrome(s, i, i + l -
1))){
//                     max = l;
//                     index[0] = i;
//                     index[1] = i + l;
//                 }
//             }else{
//                 break;
//             }
//         }
//     }

//     return s.substring(index[0], index[1]);
// }

// private boolean isPalindrome(String s, int start, int end
){
//     while(start < end){
//         if(s.charAt(start) != s.charAt(end)){
//             return false;
//         }
//         start++;
//         end--;

```

```
//    }
//    return true;
// }

//O(n^2) DP
// public String longestPalindrome(String s) {
//    // Write your code here
//    if(s == null || s.length() == 0){
//        return "";
//    }

//    int len = s.length();
//    boolean[][] dp = new boolean[len][len];

//    int max = 0;
//    String sb = "";
//    for(int j = 0; j < len; j++){
//        for(int i = 0; i <= j; i++){
//            if(s.charAt(i) == s.charAt(j)){
//                if(j - i <= 1 || dp[i + 1][j - 1]){
//                    dp[i][j] = true;
//                    if(j - i + 1 > max){
//                        max = j - i + 1;
//                        sb = s.substring(i, j + 1);
//                    }
//                }
//            }else{
//                dp[i][j] = false;
//            }
//        }
//    }

//    return sb;
// }

//O(n) 中心展开
public String longestPalindrome(String s) {
    // Write your code here
    if(s == null || s.length() == 0){
        return "";
    }
}
```

```
    }

    int len = s.length();
    int max = -1;
    String sb = "";
    for(int i = 1; i < 2 * len; i++){
        int count = 1;
        while(i - count >= 0 && i + count <= 2 * len && get(
s, i - count) == get(s, i + count)){
            count++;
        }
        count--;
        if(count > max){
            max = count;
            sb = s.substring((i - count) / 2, (i + count) /
2);
        }
    }
    return sb;
}

private char get(String s, int index){
    if(index % 2 == 0){
        return '#';
    }else{
        return s.charAt(index / 2);
    }
}
}
```

# Stone Game 476

## Question

There is a stone game. At the beginning of the game the player picks  $n$  piles of stones in a line.

The goal is to merge the stones in one pile observing the following rules:

At each step of the game, the player can merge two adjacent piles to a new pile.

The score is the number of stones in the new pile.

You are to determine the minimum of the total score.

Example

For  $[4, 1, 1, 4]$ , in the best solution, the total score is 18:

1. Merge second and third piles  $\Rightarrow [4, 2, 4]$ , score +2
2. Merge the first two piles  $\Rightarrow [6, 4]$ , score +6
3. Merge the last two piles  $\Rightarrow [10]$ , score +10

Other two examples:

$[1, 1, 1, 1]$  return 8

$[4, 4, 5, 9]$  return 43

## Solution

这道题既能用DP解，也能用记忆化搜索的方法解。

$dp[i][j]$ 表示合并 $i$ 到 $j$ 的石头需要的最小代价。

转移函数：

$dp[i][j] = dp[i][k] + dp[k+1][j] + sum[i][j]$  ( $i \leq k < j$ )。即合并 $i-j$ 的代价为合并左边部分的代价+合并右边部分的代价+合并左右部分的代价（即 $i-j$ 所有元素的总和）。找到使 $dp[i][j]$ 最小的 $k$ 。

需要初始化 $sum$ 。DP以长度和不同起点为循环条件，而记忆化搜索需要 $start$ 和 $end$ 来确定搜索范围，然后找分割点 $k$ ，再递归搜索左右部分，有点D&C的味道。

代码如下：

DP

```
public class Solution {
    /**
     * @param A an integer array
     * @return an integer
     */
    public int stoneGame(int[] A) {
        // Write your code here
        // DP
        if(A == null || A.length == 0){
            return 0;
        }

        int n = A.length;
        int[][] sum = new int[n][n];
        for(int i = 0; i < n; i++){
            sum[i][i] = A[i];
            for(int j = i + 1; j < n; j++){
                sum[i][j] = sum[i][j - 1] + A[j];
            }
        }

        int[][] dp = new int[n][n];
        for(int i = 0; i < n; i++){
            dp[i][i] = 0;
        }

        for(int len = 2; len <= n; len++){
            for(int i = 0; i + len - 1 < n; i++){
                int j = i + len - 1;
```

```
        int min = Integer.MAX_VALUE;
        for(int k = i; k < j; k++){
            min = Math.min(min, dp[i][k] + dp[k + 1][j])
;
        }
        dp[i][j] = min + sum[i][j];
    }
}

return dp[0][n - 1];
}
}
```

## Memorized Search

```
public class Solution {
    /**
     * @param A an integer array
     * @return an integer
     */
    public int stoneGame(int[] A) {
        // Write your code here
        // Memorized search
        if(A == null || A.length == 0){
            return 0;
        }

        int n = A.length;
        int[][] dp = new int[n][n];
        int[] sum = new int[n + 1];

        for(int i = 0; i < n - 1; i++){
            for(int j = i + 1; j < n; j++){
                dp[i][j] = -1;
            }
        }

        sum[0] = 0;
        for(int i = 0; i < n; i++){
```



```
        dp[i][i] = 0;
        sum[i + 1] = sum[i] + A[i];
    }

    return search(0, n - 1, sum, dp);
}

private int search(int start, int end, int[] sum, int[][] dp
){
    if(dp[start][end] >= 0){
        return dp[start][end];
    }

    int min = Integer.MAX_VALUE;
    for(int k = start; k < end; k++){
        int left = search(start, k, sum, dp);
        int right = search(k + 1, end, sum, dp);
        int now = sum[end + 1] - sum[start];
        min = Math.min(min, left + right + now);
    }
    dp[start][end] = min;
    return dp[start][end];
}
}
```

# Bomb Enemy 553

## Question

Given a 2D grid, each cell is either a wall 'W', an enemy 'E' or empty '0' (the number zero), return the maximum enemies you can kill using one bomb. The bomb kills all the enemies in the same row and column from the planted point until it hits the wall since the wall is too strong to be destroyed.

Notice

**You can only put the bomb at an empty cell.**

Example

Given a grid:

0 E 0 0

E 0 W E

0 E 0 0

return 3. (Placing a bomb at (1,1) kills 3 enemies)

## Solution

这道题的思想就是遍历数组中的每一个点，当前点能够得到的最大值为从左边的头（边界或wall）到右边的头（遇到边界或wall）的值（row值）+从上面的头（边界或wall）到下面的头（遇到边界或wall）的值（col值）。每一个row范围内的点都共用当前的row值，每一个col范围内的点都共用当前col值。当从新的边界或者wall开始时，相当于进入了新的一段范围，要重新计算row值或者col值。

1. 遍历数组中每一个点，若为0则开始计算
2. 若当前点为第一列或者左边一个点为wall，表明进入了一个新的区间，需要重新计算。从该点开始一直向右直到遇到边界或者wall，在该过程中，每遇到一个E就将row值+1

3. 若当前点为第一行或者上边一个点为wall，表明进入了一个新的区间，需要重新计算。从该点开始一直向下直到遇到边界或者wall，在该过程中，每遇到一个E就将col值+1

4. 重复2-3步骤

代码如下：

```
public class Solution {
    /**
     * @param grid Given a 2D grid, each cell is either 'W', 'E'
     * or '0'
     * @return an integer, the maximum enemies you can kill using one bomb
     */
    public int maxKilledEnemies(char[][] grid) {
        // Write your code here
        if(grid == null || grid.length == 0 || grid[0].length == 0){
            return 0;
        }

        int m = grid.length;
        int n = grid[0].length;

        int result = 0;
        int rows = 0;
        int[] cols = new int[n];
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                if (j == 0 || grid[i][j-1] == 'W') {
                    rows = 0;
                    for (int k = j; k < n && grid[i][k] != 'W'; ++k)
                        if (grid[i][k] == 'E')
                            rows += 1;
                }
                if (i == 0 || grid[i-1][j] == 'W') {
                    cols[j] = 0;
                    for (int k = i; k < m && grid[k][j] != 'W';
```

```
++k)
        if (grid[k][j] == 'E')
            cols[j] += 1;
    }

    if (grid[i][j] == '0' && rows + cols[j] > result
)
        result = rows + cols[j];
    }
}
return result;
}
}
```

# Burst Balloons 168

## Question

Given  $n$  balloons, indexed from 0 to  $n-1$ . Each balloon is painted with a number on it represented by array `nums`. You are asked to burst all the balloons. If the you burst balloon  $i$  you will get `nums[left] * nums[i] * nums[right]` coins. Here `left` and `right` are adjacent indices of  $i$ . After the burst, the `left` and `right` then becomes adjacent.

Find the maximum coins you can collect by bursting the balloons wisely.

- You may imagine `nums[-1] = nums[n] = 1`. They are not real therefore you can not burst them.
- $0 \leq n \leq 500$ ,  $0 \leq \text{nums}[i] \leq 100$

Example

Given `[4, 1, 5, 10]`

Return 270

`nums = [4, 1, 5, 10]` burst 1, get coins  $4 * 1 * 5 = 20$

`nums = [4, 5, 10]` burst 5, get coins  $4 * 5 * 10 = 200$

`nums = [4, 10]` burst 4, get coins  $1 * 4 * 10 = 40$

`nums = [10]` burst 10, get coins  $1 * 10 * 1 = 10$

Total coins  $20 + 200 + 40 + 10 = 270$

## Solution

用线段DP+记忆化搜索的方法。`DP[i][j]`表示 $i-j$ 个气球能取得的最大值。

初始化：在`nums`左右两边各增加1个dummy balloon，使其值为1。这是为了计算方便，在乘的时候不用考虑边界情况。

状态函数：假设某一个气球 $k$ 是最后打爆的，则其最大值为 $k$ 左边气球最大值+ $k$ 右边气球最大值+打爆 $k$ 的值。遍历数组，记录每个元素成为 $k$ 时能取的最大值，然后在其中找到最大的值即可。

当某个元素 $k$ 最后打爆时，其值为 $\text{array}[\text{左边界}-1] \text{array}[\text{右边界}+1] \text{array}[k]$ 。然后搜索左边界到 $k-1$ 的最大值和 $k+1$ 到右边界的最大值。

代码如下：

```
public class Solution {
    /**
     * @param nums a list of integer
     * @return an integer, maximum coins
     */
    public int maxCoins(int[] nums) {
        // Write your code here
        if(nums == null || nums.length == 0){
            return 0;
        }

        int n = nums.length;
        int[] array = new int[n + 2];
        for(int i = 1; i <= n; i++){
            array[i] = nums[i - 1];
        }
        array[0] = 1;
        array[n + 1] = 1;

        int[][] dp = new int[n + 2][n + 2];
        boolean[][] visit = new boolean[n + 2][n + 2];

        return search(1, n, array, dp, visit);
    }

    private int search(int start, int end, int[] array, int[][] dp, boolean[][] visit){
        if(visit[start][end]){
            return dp[start][end];
        }
    }
```

```
        int max = 0;
        for(int k = start; k <= end; k++){
            int midValue = array[start - 1] * array[k] * array[en
nd + 1];
            int left = search(start, k - 1, array, dp, visit);
            int right = search(k + 1, end, array, dp, visit);
            max = Math.max(max, midValue + left + right);
        }
        visit[start][end] = true;
        dp[start][end] = max;
        return dp[start][end];
    }
}
```

# Scramble String 430

## Question

Given a string  $s_1$ , we may represent it as a binary tree by partitioning it to two non-empty substrings recursively.

Below is one possible representation of  $s_1 = \text{"great"}$ :

```
great
```

```
/ \ gr eat / \ / \ g r e at / \ a t
```

To scramble the string, we may choose any non-leaf node and swap its two children.

For example, if we choose the node "gr" and swap its two children, it produces a scrambled string "rgeat".

```
rgeat
```

```
/ \ rg eat / \ / \ r g e at / \ a t
```

We say that "rgeat" is a scrambled string of "great".

Similarly, if we continue to swap the children of nodes "eat" and "at", it produces a scrambled string "rgtae".

```
rgtae
```

```
/ \ rg tae / \ / \ r g ta e / \ t a
```

We say that "rgtae" is a scrambled string of "great".

Given two strings  $s_1$  and  $s_2$  of the same length, determine if  $s_2$  is a scrambled string of  $s_1$ .



## Challenge

O(n<sup>3</sup>) time

## Solution

这道题可以用递归的方法解。对于s1和s2，找不同的分割点k，将其分别分为两个子数组。若s1的两个子数组和对应的s2的两个子数组都是scramble的，则s1和s2就是scramble的。例如，s1[0...i]被分为0...k，k+1...i，其对应的s2子数组为0...k，k+1...i或者0...i-k-1，i-k...i（即s2的前k个元素或者后k个元素对应于s1的前k个元素，比如s1=abc，s2=acb，第一层递归时比较的是s1左边的子数组和s2左边的子数组以及s1右边的子数组和s2右边的子数组，第二层递归比较右边两个子数组时，就要比较s1右边子数组的左边子数组"b"和s2右边子数组的右边子数组"b"），只要两种情况里面有一种满足，则s1和s2就是scramble的。因此：

```
( isScramble(s2[0...k], s1[0...k]) && isScramble(s2[k+1...j], s1[k+1...i]) ) || (
isScramble(s2[0...k], s1[i-k...i]) && isScramble(s2[k+1...j], s1[0...i-k-1]) ) , (k =
0,1,2 ... i-1, k相当于字符串的分割点)
```

只要一个子问题返回ture，那么就表示两个字符串可以转换。

因为递归解法有很多重复子问题，比如s2 = reat, s1 = great 当我们选择分割点为0时，要解决子问题 isScramble(reat, geat)，再对该子问题选择分割点0时，要解决子问题 isScramble(eat, eat)；而当我们第一步选择1作为分割点时，也要解决子问题 isScramble(eat, eat)。相同的子问题isScramble(eat, eat)就要解决2次。

因此可以用记忆化搜索来保存子问题，设dp[i][j][k]表示s2从j开始长度为k的子串是否可以由s1从i开始长度为k的子串转换而成，那么动态规划方程如下：

状态方程：dp[i][j][k] = ( dp[i][j][divlen] && dp[i+divlen][j+divlen][k-divlen] ) || ( dp[i][j+k-divlen][divlen] && dp[i+divlen][j][k-divlen] ) (divlen = 1,2,3...k-1, 它表示子串分割点到子串起始端的距离)，只要一个子问题返回真，就可以停止计算

代码如下：

DP:

```
public class Solution {
    /**
     * @param s1 A string
```

```
* @param s2 Another string
* @return whether s2 is a scrambled string of s1
*/
public boolean isScramble(String s1, String s2) {
    // Write your code here
    int[][][] dp = new int[n][n][n + 1];
    return checkScramble(s1, 0, s2, 0, n, dp);
}

private boolean checkScramble(String s1, int s1Start, String
s2, int s2Start, int len, int[][][] dp){
    if(dp[s1Start][s2Start][len] == 1){
        return true;
    }
    if(dp[s1Start][s2Start][len] == -1){
        return false;
    }

    if(s1.length() != s2.length()){
        return false;
    }

    if(s1.length() == 0 || s1.equals(s2)){
        dp[s1Start][s2Start][len] = 1;
        return true;
    }

    if(!isValid(s1, s2)){
        dp[s1Start][s2Start][len] = -1;
        return false;
    }

    for(int k = 1; k < len; k++){
        String s1Left = s1.substring(0, k);
        String s1Right = s1.substring(k, len);

        String s2Left = s2.substring(0, k);
        String s2Right = s2.substring(k, len);
```

```

        if(checkScramble(s1Left, s1Start, s2Left, s2Start, k
, dp) && checkScramble(s1Right, s1Start + k, s2Right, s2Start +
k, len - k, dp)){
            dp[s1Start][s2Start][len] = 1;
            return true;
        }

        s2Left = s2.substring(0, len - k);
        s2Right = s2.substring(len - k, len);

        if(checkScramble(s1Left, s1Start, s2Right, s2Start +
len - k, k, dp) && checkScramble(s1Right, s1Start + k, s2Left,
s2Start, len - k, dp)){
            dp[s1Start][s2Start][len] = 1;
            return true;
        }
    }

    dp[s1Start][s2Start][len] = -1;
    return false;
}

private boolean isValid(String s1, String s2){
    char[] array1 = s1.toCharArray();
    char[] array2 = s2.toCharArray();
    Arrays.sort(array1);
    Arrays.sort(array2);
    if(!new String(array1).equals(new String(array2))){
        return false;
    }
    return true;
}
}

```

## Recursion

```

public class Solution {
    /**

```

```
* @param s1 A string
* @param s2 Another string
* @return whether s2 is a scrambled string of s1
*/
public boolean isScramble(String s1, String s2) {
    // Write your code here
    //Recursion
    if(s1 == null && s2 == null){
        return true;
    }

    if(s1 == null || s2 == null){
        return false;
    }

    if(s1.length() != s2.length()){
        return false;
    }

    if(s1.length() == 0 || s1.equals(s2)){
        return true;
    }

    if(!isValid(s1, s2)){
        return false;
    }

    int n = s1.length();
    for(int k = 1; k < n; k++){
        String s1Left = s1.substring(0, k);
        String s1Right = s1.substring(k);
        String s2Left = s2.substring(0, k);
        String s2Right = s2.substring(k);
        if(isScramble(s1Left, s2Left) && isScramble(s1Right,
s2Right)){
            return true;
        }
        s2Left = s2.substring(0, n - k);
        s2Right = s2.substring(n - k);
        if(isScramble(s1Left, s2Right) && isScramble(s1Right,
```

```
, s2Left)){
    return true;
}
return false;
}

private boolean isValid(String s1, String s2){
    char[] array1 = s1.toCharArray();
    char[] array2 = s2.toCharArray();
    Arrays.sort(array1);
    Arrays.sort(array2);
    if(!new String(array1).equals(new String(array2))){
        return false;
    }
    return true;
}
}
```

# Copy Books 437

## Question

Given an array  $A$  of integer with size of  $n$  ( means  $n$  books and number of pages of each book) and  $k$  people to copy the book. You must distribute the continuous id books to one people to copy. (You can give book  $A[1], A[2]$  to one people, but you cannot give book  $A[1], A[3]$  to one people, because book  $A[1]$  and  $A[3]$  is not continuous.) Each person have can copy one page per minute. Return the number of smallest minutes need to copy all the books.

### Example

Given array  $A = [3, 2, 4]$ ,  $k = 2$ .

Return 5( First person spends 5 minutes to copy book 1 and book 2 and second person spends 4 minutes to copy book 3. )

### Challenge

Could you do this in  $O(n*k)$  time ?

## Solution

这道题有两种解法。第一种是基本的DP，第二种是用二分法搜索可能最小时间。

第一种DP建立一个二维数组( $n+1 * k+1$ )， $T[i][j]$ 表示前 $i$ 本书分配给 $j$ 个人copy。

1. 初始化 $T[1][j]=pages[0]$ ，初始化 $T[i][1]= pages[0] + pages[1] + ... + pages[i-1]$
2. 然后从2本书开始到 $n$ 本书为止，依次计算分配给 $2 \sim k$ 个人的最小时间。当人数比书大时，有些人不干活也不会影响速度，因此和少一个人情况相同。
3. 对于新加进来的人 $j$ ，考虑让前 $j-1$ 个人copy的书的数量 $h$  ( $0 \sim n$ )，则新进来的人相应的copy的数量为 $n \sim 0$ 本，前者的时间为 $T[h][j-1]$ ，后者的时间为 $T[i][1]-T[h][1]$ （即一个人copy从 $h+1$ 到 $i$ 本需要的时间），两者的较大值即为 $T[i][j]$ 的一个后选项。选择所有后选项中的最小值即为 $T[i][j]$ 的值。这里可以优化，即

我们知道如果前 $j-1$ 个人copy的书的数量少于 $j-1$ 必然有人不干活，而所有人都干活的结果一定会更快，所以 $h$ 的范围可以从 $j-1 \sim n$ ，因为我们知道 $h$ 为 $0 \sim j-1$ 时的结果一定不会是最优的答案。

第二种方法很巧妙地用到了二分搜索的方法。我们要找的最优解是某一个时间的临界点，当时间小于这个值时， $k$ 个人一定不可能完成任务，当时间大于等于这个值时，则可以完成。

1. 首先将时间的范围设为所有整数（ $0 \sim 999999999$ ）。计算中点作为这次的假设时间临界点，尽量让每个人的工作时间都接近这个临界点。
2. 假设当前某个人之前分配的书的页数加上当前书的页数小于当前临界点，则直接把这本书分配给这个人而不会影响最优解；
3. 若大于，则看当前书的页数是否大于临界点：1) 若大于，则说明当前的临界值太小，连这本书都不能copy完全，所以最优解一定大于当前临界点，因此要增大临界点再重复2 2) 若小于，则将书分配给下一个人copy
4. 若所有书分配完时，所需要的人数比 $k$ 小（即还剩下人没用），则说明每个人干活的时间太多，最优时间一定比当前值小，反之则说明每个人干活的时间太少，最优时间比当前值大
5. 考虑3-4中的所有情况，若最优解比当前临界点小，则向当前临界点左半边搜索，否则向当前临界点右半边搜索，直到左右边界重合，此时的临界点（即左右边界）即为最优解。

代码如下：

DP

```
public class Solution {
    /**
     * @param pages: an array of integers
     * @param k: an integer
     * @return: an integer
     */
    public int copyBooks(int[] pages, int k) {
        // write your code here
        // if(pages == null || pages.length == 0){
        //     return 0;
        // }
    }
}
```

```

        // if(k < 1){
        //     return -1;
        // }

        // int n = pages.length;
        // int[][] T = new int[n + 1][k + 1];

        // for(int j = 1; j <= k; j++){
        //     T[1][j] = pages[0];
        // }

        // int sum = 0;
        // for(int i = 1; i <= n; i++){
        //     sum += pages[i - 1];
        //     T[i][1] = sum;
        // }

        // for(int i = 2; i <= n; i++){
        //     for(int j = 2; j <= k; j++){
        //         if(j > i){
        //             T[i][j] = T[i][j - 1];
        //             continue;
        //         }
        //         int min = Integer.MAX_VALUE;
        //         for(int h = j - 1; h <= i; h++){
        //             int temp = Math.max(T[h][j - 1], T[i][1]
- T[h][1]);
        //             min = Math.min(min, temp);
        //         }
        //         T[i][j] = min;
        //     }
        // }

        // return T[n][k];
    }
}

```

Binary search



```
public class Solution {
    /**
     * @param pages: an array of integers
     * @param k: an integer
     * @return: an integer
     */
    public int copyBooks(int[] pages, int k) {
        // write your code here
        //O(n*logM)? O(n*k)
        int l = 0;
        int r = 9999999;
        while( l <= r){
            int mid = l + (r - l) / 2;
            if(search(mid,pages,k))
                r = mid-1;
            else
                l = mid+1;
        }
        return l;
    }

    private boolean search(int total, int[] page, int k){
        //至少有一个人copy，所以count从1开始
        int count = 1;
        int sum = 0;
        for(int i = 0; i < page.length; ) {
            if(sum + page[i] <= total){
                sum += page[i++];
            }else if(page[i] <= total){
                sum = 0;
                count++;
            }else{
                return false;
            }
        }

        return count <= k;
    }
}
```



# Maximum Gap 400

## Question

Given an unsorted array, find the maximum difference between the successive elements in its sorted form.

Return 0 if the array contains less than 2 elements.

Notice

You may assume all elements in the array are non-negative integers and fit in the 32-bit signed integer range.

Example

Given [1, 9, 2, 5], the sorted form of it is [1, 2, 5, 9], the maximum gap is between 5 and 9 = 4.

Challenge

Sort is easy but will cost  $O(n\log n)$  time. Try to solve it in linear time and space.

## Solution

这道题如果直接用普通排序，则可以轻易求解，但是复杂度是 $O(n\log n)$ 。题目要求 $O(n)$ 则考虑线性时间排序。这里用桶排序Bucket Sort。桶排序步骤如下：

1. 首先找出数组的最大值和最小值，特殊情况是最大值和最小值相等则说明所有值相等，直接返回0即可
2. 然后要确定每个桶的容量，即为 $(\text{最大值} - \text{最小值}) / \text{个数} + 1$ ，
3. 再确定桶的个数，即为 $(\text{最大值} - \text{最小值}) / \text{桶的容量} + 1$
4. 遍历整个数组，根据 $\text{index} = (\text{nums} - \text{min}) / \text{bucketSize}$ 将每个数加入不同的桶中，这里用两个array来记录这个桶里的最大值和最小值（因为只保留着两个值就足够了，不用记录其他的数）。用一个set来记录每个桶是否被放过数。如果

该桶内没有被放过数，则直接将该桶的最大值和最小值设为当前数，否则需要和最值进行比较后更新最值。

5. 相邻的两个数有两种情况：1) 落在同一个桶里 2) 小的那个是前一个桶的最大值大的那个是后一个桶的最小值。落在不同桶里的数之间的间距一定比落在相同桶里的数之间的间距大，因此最大间距的两个数不会在同一个桶中，而是后一个桶的最小值和前一个桶的最大值之间的间距，因此遍历两个array，用后一个桶的最大值减去前一个桶的最小值，记录最大差值即可

代码如下：

```
class Solution {
    /**
     * @param nums: an array of integers
     * @return: the maximum difference
     */
    public int maximumGap(int[] nums) {
        // write your code here
        if(nums == null || nums.length < 2){
            return 0;
        }

        // sort version
        // Arrays.sort(nums);

        // int diff = 0;
        // for(int i = 1; i < nums.length; i++){
        //     diff = Math.max(diff, nums[i] - nums[i - 1]);
        // }

        // return diff;

        //Bucket sort
        int max = Integer.MIN_VALUE;
        int min = Integer.MAX_VALUE;
        int n = nums.length;
        for(int num : nums){
            max = Math.max(max, num);
            min = Math.min(min, num);
        }
    }
}
```

```
if(max == min){
    return 0;
}

int bucketSize = (max - min) / n + 1;
int bucketNum = (max - min) / bucketSize + 1;
int[] bucketMin = new int[bucketNum];
int[] bucketMax = new int[bucketNum];
HashSet<Integer> set = new HashSet<Integer>();
for(int num : nums){
    int index = (num - min) / bucketSize;
    if(!set.contains(index)){
        bucketMin[index] = num;
        bucketMax[index] = num;
        set.add(index);
        continue;
    }

    if(bucketMin[index] > num){
        bucketMin[index] = num;
    }
    if(bucketMax[index] < num){
        bucketMax[index] = num;
    }
}
```

//一定会有数落在0bucket里，因为 $index = (num - min) / bucketSize$ ，当 $num = min$ 时就落在0桶里，所以第一个非空的桶一定为0

```
int pre = 0;
int res = 0;
//寻找下一个非空的桶，空的桶就跳过
for(int i = 1; i < bucketNum; i++){
    if(!set.contains(i)){
        continue;
    }
    res = Math.max(res, bucketMin[i] - bucketMax[pre]);
    pre = i;
}
```

```
        return res;
    }
}
```

# Longest Common Subsequence 77

## Question

Given two strings, find the longest common subsequence (LCS).

Your code should return the length of LCS.

Clarification

What's the definition of Longest Common Subsequence?

[https://en.wikipedia.org/wiki/Longest\\_common\\_subsequence\\_problem](https://en.wikipedia.org/wiki/Longest_common_subsequence_problem)

<http://baike.baidu.com/view/2020307.htm>

Example

For "ABCD" and "EDCA", the LCS is "A" (or "D", "C"), return 1.

For "ABCD" and "EACB", the LCS is "AC", return 2.

## Solution

求LCS。用DP解决。

$D[i][j]$  定义为s1, s2的前i,j个字符串的最长common subsequence.

状态函数：

当  $\text{char } i == \text{char } j$ ,  $D[i][j] = \max(D[i-1][j-1] + 1, D[i][j-1], D[i-1][j])$ , 即i可以和j匹配, 也可以和j-1匹配, 反之也一样, 三种情况里面选一个最大的。

当  $\text{char } i \neq \text{char } j$ ,  $D[i][j] = \max(D[i][j-1], D[i-1][j])$  (因为最后一个字符不相同, 所以有可能s1的最后一个字符会出现在s2的前部分里, 或者s2的最后一个字符会出现在s1的前部分里)。

代码如下：

```
public class Solution {
    /**
     * @param A, B: Two strings.
     * @return: The length of longest common subsequence of A and B.
     */
    public int longestCommonSubsequence(String A, String B) {
        // write your code here
        int n = A.length();
        int m = B.length();
        int[][] f = new int[n + 1][m + 1];
        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= m; j++){
                f[i][j] = Math.max(f[i - 1][j], f[i][j - 1]);
                if(A.charAt(i - 1) == B.charAt(j - 1)){
                    f[i][j] = Math.max(f[i - 1][j - 1] + 1, f[i]
[j]);
                }
            }
        }
        return f[n][m];
    }
}
```



# Interleaving String 29

## Question

Given three strings: s1, s2, s3, determine whether s3 is formed by the interleaving of s1 and s2.

Example

For s1 = "aabcc", s2 = "dbbca"

When s3 = "aadbcbcbac", return true.

When s3 = "aadbbaaccc", return false.

Challenge

O(n<sup>2</sup>) time or better

## Solution

dp[i][j]表示s1前i个和s2前j个对s3前i+j个是否interleaving string。

1. 首先初始化。遍历s1，初始化所有的dp[i][0]
2. 再遍历s2，初始化所有的dp[0][j]
3. 若s3的第i+j-1位和s1的第i位相等，则看dp[i-1][j]是否为true；同理，若s3的第i+j-1位和s2的第j位相等，则看dp[i][j-1]是否为true。只要两种情况中的任意一种为true，则dp[i][j]为true。

代码如下：

```
public class Solution {  
    /**  
     * Determine whether s3 is formed by interleaving of s1 and  
     * s2.  
     * @param s1, s2, s3: As description.  
     * @return: true or false.  
     */  
}
```

```
    */
    public boolean isInterleave(String s1, String s2, String s3)
    {
        // write your code here
        int n = s1.length();
        int m = s2.length();
        boolean[][] f = new boolean[n + 1][m + 1];

        if(n + m != s3.length()){
            return false;
        }
        f[0][0] = true;

        for(int i = 1; i <= n; i++){
            if(s3.charAt(i - 1) == s1.charAt(i - 1) && f[i - 1][
0]){
                f[i][0] = true;
            }
        }

        for(int j = 1; j <= m; j++){
            if(s3.charAt(j - 1) == s2.charAt(j - 1) && f[0][j -
1]){
                f[0][j] = true;
            }
        }

        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= m; j++){
                if(f[i - 1][j] && (s1.charAt(i - 1) == s3.charAt
(i + j - 1)) || f[i][j - 1] && (s2.charAt(j - 1) == s3.charAt(i
+ j - 1))){
                    f[i][j] = true;
                }
            }
        }

        return f[n][m];
    }
}
```



# Edit Distance 119

## Question

Given two words word1 and word2, find the minimum number of steps required to convert word1 to word2. (each operation is counted as 1 step.)

You have the following 3 operations permitted on a word:

Insert a character

Delete a character

Replace a character

Example

Given word1 = "mart" and word2 = "karma", return 3.

## Solution

$ed[i][j]$ 表示将word1的前i个字符变成word2的前j个字符所需要的最少操作。

初始化：

```
ed[i][0]=i 删除i次  
ed[0][j]=j 添加j次
```

状态函数：

```
如果word1.charAt(i-1)==word2.charAt(j-1)  
ed[i][j]=min(min(ed[i-1][j], ed[i][j-1])+1, ed[i-1][j-1])  
否则  
ed[i][j]=min(min(ed[i-1][j], ed[i][j-1])+1, ed[i-1][j-1]+1)
```

即word1的前i为字符转换为word2的前j位字符所需的最小操作有三种情况要讨论：

- 1) 将word1的第i位字符删去。此时只要看word1的前i-1位字符转换为word2的前j位字符所需操作再加上刚才的删除操作。
- 2) 将word2的第j位字符删去。此时只要看word1的前i位字符转换为word2的前j-1位字符所需操作再加上刚才的删除操作。
- 3) 将word1的第i位字符转换为word2的第j位字符。此时又分两种情况：1) word1的第i位字符和word2的第j位字符相等，则对于word1的第i位字符和word2的第j位字符无需任何操作，直接看word1的前i-1位字符转换为word2的前j-1位字符所需操作2) word1的第i位字符和word2的第j位字符不相等，则要将word1的第i位字符转换为word2的第j位字符，然后就是1) 的情况。

这三种情况里面的最小值就是ed[i][j]的值。

代码如下：

```
public class Solution {
    /**
     * @param word1 & word2: Two string.
     * @return: The minimum number of steps.
     */
    public int minDistance(String word1, String word2) {
        // write your code here
        if(word1 == null && word2 == null || word1.length() == 0
        && word2.length() == 0){
            return 0;
        }

        if(word1 == null || word1.length() == 0){
            return word2.length();
        }

        if(word2 == null || word2.length() == 0){
            return word1.length();
        }

        int n = word1.length();
        int m = word2.length();
        int[][] ed = new int[n + 1][m + 1];
```

```
    for(int i = 0; i <=n; i++){
        ed[i][0] = i;
    }

    for(int j = 1; j <= m; j++){
        ed[0][j] = j;
    }

    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= m; j++){
            ed[i][j] = Math.min(ed[i - 1][j], ed[i][j - 1])
+ 1;

            if(word1.charAt(i - 1) == word2.charAt(j - 1)){
                ed[i][j] = Math.min(ed[i][j], ed[i - 1][j -
1]);
            }else{
                ed[i][j] = Math.min(ed[i][j], ed[i - 1][j -
1] + 1);
            }
        }
    }

    return ed[n][m];
}
```

# Distinct Subsequences 118

## Question

Given a string  $S$  and a string  $T$ , count the number of distinct subsequences of  $T$  in  $S$ .

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ACE" is a subsequence of "ABCDE" while "AEC" is not).

Example

Given  $S = \text{"rabbbit"}\text{, } T = \text{"rabbit"}\text{, return } 3\text{.}$

Challenge

Do it in  $O(n^2)$  time and  $O(n)$  memory.

$O(n^2)$  memory is also acceptable if you do not know how to optimize memory.

## Solution

$ds[i][j]$  表示  $T$  的前  $j$  个字符能够在  $S$  的前  $i$  个字符中能够取的子字符串的数量。

初始化：

```
ds[i][0]=1，即T的前0个字符总是能够成为S的一个子字符串
ds[0][j]=0(j>0)，即不可能有非空字符串是S的前0个字符的子字符串
```

状态函数：

```
如果S.charAt(i-1)==T.charAt(j-1):  
ds[i][j]=ds[i-1][j]+ds[i-1][j-1]  
否则:  
ds[i][j]=ds[i-1][j]
```

即对于S的第i位字符有两种情况：第一种为不能和T的j位字符匹配，则其值为S的前i-1位字符和T的前j位字符的匹配数；第二种情况为可以和T的第j位字符匹配，则其值为S的前i-1位字符和T的前j位字符的匹配数+S的前i-1位字符和T的前j-1位字符的匹配数。

代码如下：



```
public class Solution {
    /**
     * @param S, T: Two string.
     * @return: Count the number of distinct subsequences
     */
    public int numDistinct(String S, String T) {
        // write your code here
        if(S == null || T == null){
            return 0;
        }

        int n = S.length();
        int m = T.length();
        int[][] ds = new int[n + 1][m + 1];

        for(int i = 0; i <= n; i++){
            ds[i][0] = 1;
        }

        for(int j = 1; j <= m; j++){
            ds[0][j] = 0;
        }

        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= m; j++){
                ds[i][j] = ds[i - 1][j];
                if(S.charAt(i - 1) == T.charAt(j - 1)){
                    ds[i][j] += ds[i - 1][j - 1];
                }
            }
        }

        return ds[n][m];
    }
}
```

# Dices Sum 20

## Question

Throw  $n$  dices, the sum of the dices' faces is  $S$ . Given  $n$ , find the all possible value of  $S$  along with its probability.

Notice

You do not care about the accuracy of the result, we will help you to output results.

Example

Given  $n = 1$ , return  $[[1, 0.17], [2, 0.17], [3, 0.17], [4, 0.17], [5, 0.17], [6, 0.17]]$ .

## Solution

用dp解决。dp[i][j]表示i个骰子一共得到j点的概率。要得到dp[i][j]可以考虑若最后一个筛子的点数为k（1~6），则前i-1个筛子一共得到的点数为j-k（因为i-1个筛子至少得到i-1点，所以 $j-k \geq i-1 \Rightarrow k \leq j-i+1$ ）。所以只要把最后一个筛子为k的各种情况加起来最后再除以6即可（每多一个筛子概率要多除以一个6）。

状态函数：

```
dp[i][j] = dp[i-1][j-k] (i <= j <= 6 * i, k >= 1 && k <= j - i + 1 && k <= 6)
```

代码如下：

```
public class Solution {  
    /**  
     * @param n an integer  
     * @return a list of Map.Entry<sum, probability>  
     */  
    public List<Map.Entry<Integer, Double>> dicesSum(int n) {
```

```

        // Write your code here
        // Ps. new AbstractMap.SimpleEntry<Integer, Double>(sum,
pro)
        // to create the pair
        List<Map.Entry<Integer, Double>> result = new ArrayList<
Map.Entry<Integer, Double>>();
        if(n < 1){
            return result;
        }
        //初始化n=1的情况
        double[][] matrix = new double[n + 1][6 * n + 1];
        for(int i = 1; i <= 6; i++){
            matrix[1][i] = 1.0/6;
        }

        for(int i = 2; i <= n; i++){
            //i个筛子至少得到i点，至多得到6 * i点
            for(int j = i; j <= 6 * i; j++){
                //k表示最后一个筛子能取的点数
                for(int k = 1; k <= 6; k++){
                    if(k <= j - i + 1){
                        matrix[i][j] += matrix[i - 1][j - k];
                    }
                }
                //相对i-1个筛子多了一个筛子，因此加和的每一项都要除以6
                matrix[i][j] /= 6.0;
            }
        }

        for(int i = n; i <= 6 * n; i++){
            result.add(new AbstractMap.SimpleEntry<Integer, Double>(i, matrix[n][i]));
        }

        return result;
    }
}

```



# Maximum Subarray III 43

## Question

Given an array of integers and a number  $k$ , find  $k$  non-overlapping subarrays which have the largest sum.

The number in each subarray should be contiguous.

Return the largest sum.

Notice

The subarray should contain at least one number

Example

Given  $[-1, 4, -2, 3, -2, 3]$ ,  $k=2$ , return 8

## Solution

$local[i][k]$ 表示前 $i$ 个元素取 $k$ 个子数组并且必须包含第 $i$ 个元素的 $最大和$ 。

$global[i][k]$ 表示前 $i$ 个元素取 $k$ 个子数组不一定包含第 $i$ 个元素的 $最大和$ 。

$local[i][k]$ 的状态函数：

```
max(global[i-1][k-1], local[i-1][k]) + nums[i-1]
```

有两种情况，第一种是第 $i$ 个元素自己组成一个子数组，则要在前 $i-1$ 个元素中找 $k-1$ 个子数组，第二种情况是第 $i$ 个元素属于前一个元素的子数组，因此要在 $i-1$ 个元素中找 $k$ 个子数组（并且必须包含第 $i-1$ 个元素，这样第 $i$ 个元素才能合并到最后一个子数组中），取两种情况里面大的那个。

$global[i][k]$ 的状态函数：

```
max(global[i-1][k], local[i][k])
```

有两种情况，第一种是不包含第 $i$ 个元素，所以要在前 $i-1$ 个元素中找 $k$ 个子数组，第二种情况为包含第 $i$ 个元素，在 $i$ 个元素中找 $k$ 个子数组且必须包含第 $i$ 个元素，取两种情况里面大的那个。

代码如下：

```
public class Solution {
    /**
     * @param nums: A list of integers
     * @param k: An integer denote to find k non-overlapping sub
     arrays
     * @return: An integer denote the sum of max k non-overlappi
     ng subarrays
     */
    public int maxSubArray(int[] nums, int k) {
        // write your code here
        if(nums.length < k){
            return 0;
        }

        int len = nums.length;

        //local[i][k]表示前i个元素取k个子数组并且必须包含第i个元素的最大
        和。

        int[][] localMax = new int[len + 1][k + 1];
        //global[i][k]表示前i个元素取k个子数组不一定包含第i个元素的最大
        和。

        int[][] globalMax = new int[len + 1][k + 1];

        for(int j = 1; j <= k; j++){
            //前j-1个元素不可能找到不重叠的j个子数组，因此初始化为最小值，以
            防后面被取到

            localMax[j - 1][j] = Integer.MIN_VALUE;
            for(int i = j; i <= len; i++){
                localMax[i][j] = Math.max(globalMax[i - 1][j - 1
                ], localMax[i - 1][j]) + nums[i - 1];
                if(i == j){
                    globalMax[i][j] = localMax[i][j];
                }else{
                    globalMax[i][j] = Math.max(globalMax[i - 1][
```

```
j], localMax[i][j]);  
        }  
    }  
    return globalMax[len][k];  
}
```

# Best Time to Buy and Sell Stock IV 393

## Question

Say you have an array for which the  $i$ th element is the price of a given stock on day  $i$ .

Design an algorithm to find the maximum profit. You may complete at most  $k$  transactions.

Notice

You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

Example

Given prices = [4,4,6,1,1,4,2,5], and  $k = 2$ , return 6.

Challenge

$O(nk)$  time.

## Solution

如果 $k$ 一半元素的数量 $prices.length/2$ 还大，则可以用贪心算法，把后一天比前一天大的值都加进来。最坏的情况是一天比前一天大，一天比前一天小，这样最多也就是交易 $prices.length/2$ 次。

否则就要用dp。思想和Maximum Subarray III一样。 $mustSell[i][j]$ 表示前 $i$ 天最多进行 $j$ 次交易并且第 $i$ 天必须卖出的最大值。 $globalBest[i][j]$ 表示前 $i$ 天最多进行 $j$ 次交易且第 $i$ 天不一定要卖出的最大值。

初始化： $mustSell[1][j]=globalBest[1][j]=0$ ，即不管交易多少次，前1天买进后卖出所得利润最大为0。

状态函数：



```
mustSell[i][j]=max(globalBest[i-1][j-1], mustSell[i-1][j])
```

第*i*天必须交易能获得的最大值分两种情况讨论：1) 前*i-1*天交易*j-1*次，再加上第*i*天的1次交易，正好是前*i*天交易*j*次。2) 前*i-1*天交易*j*次并且第*i-1*天必须交易，则第*i*天的交易可以合并到最后一次交易中，即第*i-1*天卖出之后再买入，然后再第*i*天卖出的结果和在第*i-1*天不交易而直接在第*i*天交易的结果是一样的，因此两次交易可以合并，所以合并后这种情况也是前*i*天交易*j*次。这两种情况里面取大的为结果。

```
globalBest[i][j]=max(globalBest[i-1][j], mustSell[i][j])
```

第*i*天不一定要交易的最大值分两种情况讨论：1) 第*i*天一定交易的最大值，即上面所求的mustSell[i][j]。2) 第*i*天一定不交易的最大值，因为第*i*天确定不交易，所以只要看前*i-1*天的结果就可以了，即globalBest[i-1][j]。两个里面取大的。

代码如下：

```
class Solution {
    /**
     * @param k: An integer
     * @param prices: Given an integer array
     * @return: Maximum profit
     */
    public int maxProfit(int k, int[] prices) {
        // write your code here
        // if(k <= 0){
        //     return 0;
        // }
        //greedy
        if(k >= prices.length / 2){
            int profit = 0;
            for(int i = 1; i < prices.length; i++){
                if(prices[i] > prices[i - 1]){
                    profit += prices[i] - prices[i - 1];
                }
            }
            return profit;
        }
    }
}
```

```
    }

    int n = prices.length;
    int[][] mustsell = new int[n + 1][k + 1];
    int[][] globalbest = new int[n + 1][k + 1];

    for(int j = 0; j <= k; j++){
        mustsell[1][j] = globalbest[1][j] = 0;
    }

    for(int i = 2; i <= n; i++){
        int gainorlose = prices[i - 1] - prices[i - 2];
        mustsell[i][0] = globalbest[i][0] = 0;
        for(int j = 1; j <= k; j++){
            mustsell[i][j] = Math.max(globalbest[i - 1][j - 1], mustsell[i - 1][j]) + gainorlose;
            globalbest[i][j] = Math.max(globalbest[i - 1][j], mustsell[i][j]);
        }
    }

    return globalbest[n][k];
}
};
```

## k Sum 89

### Question

Given  $n$  distinct positive integers, integer  $k$  ( $k \leq n$ ) and a number target.

Find  $k$  numbers where sum is target. Calculate how many solutions there are?

Example

Given  $[1,2,3,4]$ ,  $k = 2$ , target = 5.

There are 2 solutions:  $[1,4]$  and  $[2,3]$ .

Return 2.

### Solution

用三维动态规划。

$ksum[i][j][l]$  表示前  $j$  个元素里面取  $l$  个元素之和为  $i$ 。

初始化： $ksum[0][j][0] = 1$  ( $j: 0 \sim n$ )，即前  $j$  个元素里面取 0 个元素使其和为 0 的方法只有 1 种，那就是什么都不取

状态函数：

$$ksum[i][j][l] = ksum[i][j-1][l] + ksum[i-A[i-1]][j-1][l-1]$$

即前  $j$  个元素里面取  $l$  个元素之和为  $i$  由两种情况组成：第一种情况为不包含第  $i$  个元素，即前  $j-1$  个元素里取  $l$  个元素使其和为  $i$ ，第二种情况为包含第  $i$  个元素，即前  $j-1$  个元素里取  $l-1$  个元素使其和为  $i-A[i-1]$ （前提是  $i-A[i-1] \geq 0$ ）。

代码如下：

```

public class Solution {
    /**
     * @param A: an integer array.
     * @param k: a positive integer (k <= length(A))
     * @param target: a integer
     * @return an integer
     */
    public int kSum(int A[], int k, int target) {
        // write your code here
        if(A == null || A.length == 0){
            return 0;
        }

        int n = A.length;
        int[][][] ksum = new int[target + 1][n + 1][k + 1];

        for(int j = 0; j <= n; j++){
            ksum[0][j][0] = 1;
        }

        for(int i = 1; i <= target; i++){
            for(int j = 1; j <= n; j++){
                for(int l = 1; l <= j && l <= k; l++){
                    ksum[i][j][l] = ksum[i][j - 1][l];
                    if(i - A[j - 1] >= 0){
                        ksum[i][j][l] += ksum[i - A[j - 1]][j -
1][l - 1];
                    }
                }
            }
        }

        return ksum[target][n][k];
    }
}

```

# Minimum Adjustment Cost 91

## Question

Given an integer array, adjust each integers so that the difference of every adjacent integers are not greater than a given number target.

If the array before adjustment is A, the array after adjustment is B, you should minimize the sum of  $|A[i]-B[i]|$

Notice

You can assume each number in the array is a positive integer and not greater than 100.

Example

Given [1,4,2,3] and target = 1, one of the solutions is [2,3,2,3], the adjustment cost is 2 and it's minimal.

Return 2.

## Solution

$mac[i][j]$ 表示前*i*个元素满足要求且第*i*个元素为*j*的最小cost。

初始化： $mac[1][j] = Math.abs(A[0] - j)$ ，根据题意*j*在1到100之间

状态函数：

```
mac[i][j] = min(max[i][j], mac[i-1][k] + abs(A[i-1] - j))
```

对于所有在范围内的*k*，当第*i*位元素取*j*时，取符合要求的第*i*—1位元素为*k*的情况的最小值，其中 $abs(j-k)$ 的值要小于target才能符合要求。

代码如下：

```
public class Solution {
```

```

/**
 * @param A: An integer array.
 * @param target: An integer.
 */
public int MinAdjustmentCost(ArrayList<Integer> A, int target) {
    // write your code here
    if(A == null || A.size() == 0){
        return 0;
    }

    int n = A.size();
    int[][] mac = new int[n + 1][100 + 1];

    for(int i = 1; i <= 100; i++){
        mac[1][i] = Math.abs(A.get(0) - i);
    }

    for(int i = 2; i <= n; i++){
        for(int j = 1; j <= 100; j++){

            mac[i][j] = Integer.MAX_VALUE;

            for(int k = 1; k <= 100; k++){
                if(Math.abs(k - j) <= target){
                    mac[i][j] = Math.min(mac[i][j], mac[i - 1][k] + Math.abs(A.get(i - 1) - j));
                }
            }
        }
    }

    int min = Integer.MAX_VALUE;
    for(int i = 1; i <= 100; i++){
        min = Math.min(min, mac[n][i]);
    }

    return min;
}

```



# Binary Tree Maximum Path Sum 94

## Question

Given a binary tree, find the maximum path sum.

The path may start and end at any node in the tree.

Example

Given the below binary tree:

1 / \ 2 3

return 6.

## Solution

这题严格意义上应该属于D&C。singlePath表示以该点为root的子树从root出发能够取的最大path，maxPath表示以该点为root的子树中任意一点到任意一点能取的最大path。

对于每一个node，以该点为root的子树的最大path有两种可能：第一种为不经过该root的path，第二种为经过该root的path。对于第一种情况，最大path为root左边或者右边的最大path，第二种为左边的singlePath + 右边的singlePath + root值。

singlePath的状态函数： $\max(\max(\text{root.left.singlePath}, \text{root.rigth.singlePath}) + \text{root.val}, 0)$ ，表示若结果为负数，则可以不包含任何点

maxPathd的状态函数： $\max(\text{root.left.maxPath}, \text{root.right.maxPath}, \text{root.left.singlePath} + \text{root.right.singlePath} + \text{root.val})$ ，至少包含一个点。

代码如下：

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
```



```

*     public TreeNode left, right;
*     public TreeNode(int val) {
*         this.val = val;
*         this.left = this.right = null;
*     }
* }
*/
class ResultType{
    int singlePath;
    int maxPath;
    public ResultType(int singlePath, int maxPath){
        this.singlePath = singlePath;
        this.maxPath = maxPath;
    }
}

public class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: An integer.
     */
    public int maxPathSum(TreeNode root) {
        // write your code here
        ResultType r = helper(root);
        return r.maxPath;
    }

    public ResultType helper(TreeNode root){
        if(root == null){
            return new ResultType(0, Integer.MIN_VALUE);
        }

        ResultType left = helper(root.left);
        ResultType right = helper(root.right);
        //singlePath不一定至少包含一个node
        int s = Math.max(Math.max(left.singlePath, right.singleP
ath) + root.val, 0);

        int m = Math.max(left.maxPath, right.maxPath);
        m = Math.max(m, left.singlePath + right.singlePath + roo

```

```
t.val);  
  
    return new ResultType(s, m);  
}  
}
```

# Word Break 107

## Question

Given a string *s* and a dictionary of words *dict*, determine if *s* can be break into a space-separated sequence of one or more dictionary words.

Example

Given *s* = "lintcode", *dict* = ["lint", "code"].

Return true because "lintcode" can be break as "lint code".

## Solution

*wb[i]*表示前*i*个字符能否组成*dict*中的word。在*i*之前寻找分割点*j*，若前*j*个字符的结果为true并且*j-i*的字符串在*dict*中，则*wb[i]*为true。

状态函数：*wb[i] = wb[i-j] && dict.contains(s.substring(i-j, i))*，*j*表示最后一个word的长度。即前*i*个字符能否组成*dict*中的word取决于前*i-j*个字符能否组成*dict*中的word以及最后一个字符串能否组成*dict*中的word。

其中，可以先遍历*dict*得到最长字符串的长度，然后将*j*限制在该长度内，可以优化时间。

代码如下：

```
public class Solution {
    /**
     * @param s: A string s
     * @param dict: A dictionary of words dict
     */
    private int getMaxLength(Set<String> dict){
        int maxLength = 0;
        for(String curt : dict){
            if(curt.length() > maxLength){
                maxLength = curt.length();
            }
        }
    }
}
```

```
        }
    }
    return maxLength;
}

public boolean wordBreak(String s, Set<String> dict) {
    // write your code here
    if(s == null){
        return false;
    }

    int n = s.length();
    int maxLength = getMaxLength(dict);
    boolean[] wb = new boolean[n + 1];
    wb[0] = true;

    for(int i = 1; i <= n; i++){
        for(int lastLength = 1; lastLength <= maxLength && lastLength <= i; lastLength++){
            if(wb[i - lastLength]){
                String last = s.substring(i - lastLength, i);

                if(dict.contains(last)){
                    wb[i] = true;
                    break;
                }
            }
        }
    }

    return wb[n];
}
```

# Palindrome Partitioning II 108

## Question

Given a string *s*, cut *s* into some substrings such that every substring is a palindrome.

Return the minimum cuts needed for a palindrome partitioning of *s*.

Example

Given *s* = "aab",

Return 1 since the palindrome partitioning ["aa", "b"] could be produced using 1 cut.

## Solution

两次动态规划。第一次动态规划`palindrome[i][j]`，表示*s*中*i~j*的字符串是否是回文串。第二次动态规划根据`palindrome`求出`cut[i]`，即前*i*个字符最少要切几次才能将字符串切为回文串。

`palindrome`：

初始化：

```
palindrome[i][i]=true，palindrome[i][i+1] = (s.charAt(i) == s.charAt(i+1))
```

即初始化一个字符和两个字符的字符串。

状态函数：`palindrome[i][j] = s.charAt(i) == s.charAt(j) && palindrome[i+1][j-1]`，即两头字符要相等，同时中间字符串也要为回文串。

`cut`:

初始化：`cut[i]=i-1`，即前*i*个元素至多需要切*i-1*次,这里特别要注意`cut[0]=-1`，这是为了后面状态函数中*j=0*时能够满足题意。

状态函数： $cut[i]=\min(cut[i], cut[j]+1)$ ，即在 $i$ 之前的位置上找一个切割点 $j$ ，使得 $j\sim i$ 的字符串为回文串（ $palindrome[j][i-1]=true$ ），这样将前 $i$ 个字符串切成回文串需要的次数为将前 $j$ 个字符串切成回文串需要的最小次数+1（1表示切 $j$ 的这一次），取不同 $j$ 情况下的最小值即可。注意， $i$ （前 $i$ 个字符）和 $i-1$ （第 $i-1$ 个字符）的含义和转换。

代码如下：

```
public class Solution {
    /**
     * @param s a string
     * @return an integer
     */
    private boolean[][] isPalindrome(String s){
        int m = s.length();
        boolean[][] result = new boolean[m][m];

        for(int i = 0; i < m; i++){
            result[i][i] = true;
        }

        for(int i = 0; i < m - 1; i++){
            result[i][i + 1] = (s.charAt(i) == s.charAt(i + 1));
        }

        for(int i = 2; i < m; i++){
            for(int j = 0; j < i - 1; j++){
                if(s.charAt(j) == s.charAt(i) && result[j + 1][i
- 1]){
                    result[j][i] = true;
                }
            }
        }

        return result;
    }

    public int minCut(String s) {
        // write your code here
        if(s == null || s.length() == 0){
```

```

        return 0;
    }

    int n = s.length();
    int[] cut = new int[n + 1];
    boolean[][] palindrome = isPalindrome(s);

    for(int i = 0; i <= n; i++){
        cut[i] = i - 1;
    }

    for(int i = 1; i <= n; i++){
        for(int j = 0; j < i; j++){
            if(palindrome[j][i - 1]){
                cut[i] = Math.min(cut[i], cut[j] + 1);
            }
        }
    }

    return cut[n];
}
};

```

# Triangle 109

## Question

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

Notice

Bonus point if you are able to do this using only  $O(n)$  extra space, where  $n$  is the total number of rows in the triangle.

Example

Given the following triangle:

[ [2], [3,4], [6,5,7], [4,1,8,3] ]

The minimum path sum from top to bottom is 11 (i.e.,  $2 + 3 + 5 + 1 = 11$ ).

## Solution

$f[i][j]$ 表示从 $(i,j)$ 出发的path能取的最小值。

初始化： $f[n-1][j]=\text{triangle}[n-1][j]$ ，即最底部的点出发到达最低部的最小path为本身的值。

状态函数： $f[i][j]=\text{triangle}[i][j]+\min(f[i+1][j], f[i+1][j+1])$ ，即从 $(i,j)$ 出发的最小path为该点的值 $\text{triangle}[i][j]$ +下一行中和该点相邻的点的最小path。

优化：可以用滚动数组将 $f[n][n]$ 优化为 $f[2][n]$ ，因为 $f[i][j]$ 只和下一行的值有关，因此可以只记录下一行的值和当前行的值，将真实的行数 $i\%2$ 之后即为在滚动数组中的行数。

代码如下：



```
public class Solution {
    /**
     * @param triangle: a list of lists of integers.
     * @return: An integer, minimum path sum.
     */
    public int minimumTotal(int[][] triangle) {
        // write your code here
        if(triangle == null || triangle.length == 0){
            return 0;
        }

        if(triangle[0] == null || triangle[0].length == 0){
            return 0;
        }

        int n = triangle.length;
        int[][] f = new int[2][n];
        for(int i = 0; i < n; i++){
            f[(n - 1) % 2][i] = triangle[n - 1][i];
        }

        for(int i = n - 2; i >= 0; i--){
            for(int j = 0; j <= i; j++){
                f[i % 2][j] = triangle[i][j] + Math.min(f[(i + 1) % 2][j], f[(i + 1) % 2][j + 1]);
            }
        }

        return f[0][0];
    }
}
```

# Minimum Path Sum 110

## Question

Given a  $m \times n$  grid filled with non-negative numbers, find a path from top left to bottom right which minimizes the sum of all numbers along its path.

Notice

You can only move either down or right at any point in time.

## Solution

这题是机器人从左上走到右下的最小path的变体。

$pathSum[i][j]$ 表示从 $(0,0)$ 到 $(i,j)$ 的最小path。

初始化：

```
pathSum[i][0]=grid[i][0]+pathSum[i-1][0]
pathSum[0][j]=grid[0][j]+pathSum[0][j-1]
```

即初始化第一行和第一列

状态函数：

```
pathSum[i][j] = grid[i][j]+min(pathSum[i][j-1], pathSum[i-1][j])
```

即从 $(0,0)$ 到 $(i,j)$ 的最小path为从 $(0,0)$ 到其左边格子和上面格子的最小path的较小者+ $(i,j)$ 的值。

代码如下：

```
public class Solution {
    /**
     * @param grid: a list of lists of integers.
```

```
* @return: An integer, minimizes the sum of all numbers along its path
*/
public int minPathSum(int[][] grid) {
    // write your code here
    if(grid == null || grid.length == 0){
        return 0;
    }

    if(grid[0] == null || grid[0].length == 0){
        return 0;
    }

    int m = grid.length;
    int n = grid[0].length;
    int[][] pathSum = new int[m][n];
    pathSum[0][0] = grid[0][0];

    for(int i = 1; i < m; i++){
        pathSum[i][0] = pathSum[i - 1][0] + grid[i][0];
    }

    for(int j = 1; j < n; j++){
        pathSum[0][j] = pathSum[0][j - 1] + grid[0][j];
    }

    for(int i = 1; i < m; i++){
        for(int j = 1; j < n; j++){
            pathSum[i][j] = Math.min(pathSum[i][j - 1], pathSum[i - 1][j]) + grid[i][j];
        }
    }

    return pathSum[m - 1][n - 1];
}
```

# Climbing Stairs 111

## Question

You are climbing a stair case. It takes  $n$  steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example

Given an example  $n=3$  ,  $1+1+1=2+1=1+2=3$

return 3

## Solution

$ways[i]$ 表示到 $i$ 有多少种方法。

初始化： $ways[0]=1$ ,  $ways[1]=1$

状态函数： $ways[i]=ways[i-1]+ways[i-2]$ ，即到 $i$ 的方法数为到 $i-1$ 的方法数+到 $i-2$ 的方法数。

优化：动态数组。

代码如下：

```
public class Solution {  
    /**  
     * @param n: An integer  
     * @return: An integer  
     */  
    public int climbStairs(int n) {  
        // write your code here  
        if(n <= 1){  
            return 1;  
        }  
  
        int[] ways = new int[2];  
        ways[0] = 1;  
        ways[1] = 1;  
  
        for(int i = 2; i < n + 1; i++){  
            ways[i%2] = ways[(i - 1)%2] + ways[(i - 2)%2];  
        }  
  
        return ways[n%2];  
    }  
}
```

# Jump Game 116

## Question

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

Notice

This problem have two method which is Greedy and Dynamic Programming.

The time complexity of Greedy method is  $O(n)$ .

The time complexity of Dynamic Programming method is  $O(n^2)$ .

We manually set the small data set to allow you pass the test in both ways. This is just to let you learn how to use this problem in dynamic programming ways. If you finish it in dynamic programming ways, you can try greedy method to make it accept again.

Example

$A = [2, 3, 1, 1, 4]$ , return true.

$A = [3, 2, 1, 0, 4]$ , return false.

## Solution

DP:  $can[i]$ 表示能否到达 $i$ 点。初始化 $can[0]=true$ 。对于每个点 $i$ ，扫描其之前的所有点 $j$ ，若 $j$ 是可到达的 ( $can[j]=true$ )并且 $A[j]+j \geq i$ 则 $i$ 也是可到达的。

Greedy: 用 $farthest$ 来记录能够到达的最远的点。扫描最远的点之前的所有点，如果 $A[j]+j > farthest$ ，则更新最远的点。最后看最远的点是否能到达数组尾部。

代码如下：

## DP

```
public class Solution {  
    /**  
     * @param A: A list of integers  
     * @return: The boolean answer  
     */  
    public boolean canJump(int[] A) {  
        // wirte your code here  
        if(A == null || A.length == 0){  
            return false;  
        }  
  
        boolean[] can = new boolean[A.length];  
        can[0] = true;  
  
        for(int i = 1; i < A.length; i++){  
            for(int j = 0; j < i; j++){  
                if(can[j] == true && A[j] + j >= i){  
                    can[i] = true;  
                    break;  
                }  
            }  
        }  
  
        return can[A.length - 1];  
    }  
}
```

## Greedy

```
public class Solution {  
    /**  
     * @param A: A list of integers  
     * @return: The boolean answer  
     */  
    public boolean canJump(int[] A) {  
        // wirte your code here  
        if(A == null || A.length == 0){  
            return false;  
        }  
  
        int farthest = A[0];  
        for(int i =1; i < A.length; i++){  
            if(i <= farthest && A[i] + i > farthest){  
                farthest = A[i] + i;  
            }  
        }  
  
        return farthest >= A.length - 1? true : false;  
    }  
}
```



# Jump Game II 117

## Question

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

For example:

Given array A = [2,3,1,1,4]

The minimum number of jumps to reach the last index is 2. (Jump 1 step from index 0 to 1, then 3 steps to the last index.)

Note:

You can assume that you can always reach the last index.

## Solution

代码如下：

```
public class Solution {
    public int jump(int[] nums) {
        if(nums == null || nums.length == 0){
            return 0;
        }

        int[] minJump = new int[nums.length];
        minJump[0] = 0;
        for(int i = 1; i < nums.length; i++){
            for(int j = 0; j < i; j++){
                if(nums[j] + j >= i){
                    minJump[i] = minJump[j] + 1;
                    break;
                }
            }
        }
        return minJump[nums.length - 1];
    }
}
```

# Maximum Product Subarray 191

## Question

Find the contiguous subarray within an array (containing at least one number) which has the largest product.

Example

For example, given the array [2,3,-2,4], the contiguous subarray [2,3] has the largest product = 6.

## Solution

prod[i]记录了以第i个元素结尾的最大和最小product。

状态函数：

```
prod[i].max=max(max(prod[i-1].max*nums[i], prod[i-1].min*nums[i]), nums[i])
prod[i].min=min(min(prod[i-1].max*nums[i], prod[i-1].min*nums[i]), nums[i])
```

因为不确定当前元素的正负，所以要和之前的max和min都相乘，较大者和当前元素比较求max，较小者和当前元素比较求min。

也可以分情况讨论当前元素的正负，以确定求max和min的时候是和之前的max相乘还是和min相乘。

代码如下：

```
class Pair{
    int max;
    int min;
    public Pair(int max, int min){
        this.max = max;
        this.min = min;
    }
}

public class Solution {
    /**
     * @param nums: an array of integers
     * @return: an integer
     */
    public int maxProduct(int[] nums) {
        // write your code here
        if(nums == null || nums.length == 0){
            return 0;
        }

        Pair[] prod = new Pair[nums.length];
        prod[0] = new Pair(nums[0], nums[0]);

        int maxProd = prod[0].max;
        for(int i = 1; i < nums.length; i++){
            int max = Math.max(Math.max(prod[i - 1].max * nums[i], prod[i - 1].min * nums[i]), nums[i]);
            int min = Math.min(Math.min(prod[i - 1].max * nums[i], prod[i - 1].min * nums[i]), nums[i]);
            prod[i] = new Pair(max, min);
            maxProd = Math.max(maxProd, prod[i].max);
        }

        return maxProd;
    }
}
```

分情况讨论version

```
public class Solution {  
    /**  
     * @param nums: an array of integers  
     * @return: an integer  
     */  
    public int maxProduct(int[] nums) {  
        int[] max = new int[nums.length];  
        int[] min = new int[nums.length];  
  
        min[0] = max[0] = nums[0];  
        int result = nums[0];  
        for (int i = 1; i < nums.length; i++) {  
            min[i] = max[i] = nums[i];  
            if (nums[i] > 0) {  
                max[i] = Math.max(max[i], max[i - 1] * nums[i]);  
                min[i] = Math.min(min[i], min[i - 1] * nums[i]);  
            } else if (nums[i] < 0) {  
                max[i] = Math.max(max[i], min[i - 1] * nums[i]);  
                min[i] = Math.min(min[i], max[i - 1] * nums[i]);  
            }  
  
            result = Math.max(result, max[i]);  
        }  
  
        return result;  
    }  
}
```

# Wildcard Matching 192

## Question

Implement wildcard pattern matching with support for '?' and '\*'.

'?' Matches any single character.

'\*' Matches any sequence of characters (including the empty sequence).

The matching should cover the entire input string (not partial).

Example

`isMatch("aa","a") → false`

`isMatch("aa","aa") → true`

`isMatch("aaa","aa") → false`

`isMatch("aa","*") → true`

`isMatch("aa","a*") → true`

`isMatch("ab","?*") → true`

`isMatch("aab","cab") → false`

## Solution

DP解法：

`result[i][j]`表示S的前i个字符能否和T的前j个字符匹配。

初始化：`result[0][j]=result[0][j-1] && T.charAt(j-1)==''`，即只有“才可能和空字符匹配

状态函数：

```

if T.charAt(j-1)=='?' || S.charAt(i-1)==T.charAt(j-1)
result[i][j]=result[i-1][j-1]
if T.charAt(j-1)=='*'
result[i][j]=result[i-1][j] || result[i][j-1]

```

情况1.  $T(j) == ? \parallel T(j) == S(i)$  这时候  $f(i, j) = f(i - 1, j - 1)$ 。就是说 $T(j)$ 必须去匹配，不然这个字母没意义了。

情况2.  $T(j) == *$ ，此时分为两种情况。第一种情况让 $T(j)$ 的和 $S(i)$ 匹配，所以看 $result[i - 1][j]$ 的值（ $j$ 匹配完还在因为可以匹配任意字符；第二种情况不让 $T(j)$ 和 $S(i)$ 匹配（也可以看成匹配为空字符），此时看 $result[i][j - 1]$ 的值。两种情况只要有只要有之只要有一种为true，则 $result[i][j]$ 则责=true。

双指针解法：

实质是贪心算法： whenever encounter " in p, keep record of the current position of " in p and the current index in s. Try to match the stuff behind this "\*" in p with s, if not matched, just s++ and then try to match again.

指针star记录p中遇见的最后一个" "的位置（前面的" "如果没有和s中字符匹配可以认为已经和" "匹配），指针mark记录s中第一个需要和" \*"匹配的字符的位置。

两比较字符串		i	i	i										
	d	b	c	e	f	g	s	d	f	h	j			
	d	b	*	f	e									
				j										
	i和j就是保留位置				继续比较，c f不匹配，那么i+1，j不用加1									
					继续比较e f还不相等i+1									
					比较f和f相等									

1. 开始遍历s和p，如果两个字符匹配（p中字符和s中字符相同或者p中字符为"?"），则i和j同时前进一位。
2. 如果p中字符为" "，则用star记下该位置，用mark记录下此时i的位置，同时j前进一位。可以认为该" "被保留，如果有需要之后可以被拿出来和s中从i位置开始的任意字符进行匹配。
3. 如果p中字符不为1和2中的情况，表明p中字符和s中字符无法匹配，此时需要看是否之前有保留的" "可以使用，如果没有就gg，如果有就将该" "和s中mark位置字符进行匹配，同时将j放回该" "位置后面一位，将mark前进一位，同时

将*i*放到此时*mark*的位置。""继续保留，因为"\*"可以和任意长度字符匹配，所以在这次匹配之后不会被消耗。

4. 最后如果*s*遍历完成，而*p*中还有字符剩余，则看*p*中剩余字符是否都为"\*"

代码如下：

DP version：

```
public class Solution {
    /**
     * @param s: A string
     * @param p: A string includes "?" and "*"
     * @return: A boolean
     */
    public boolean isMatch(String s, String p) {
        // write your code here
        if(s.length() == 0){
            for(int i = 0; i < p.length(); i++){
                if(p.charAt(i) != '*'){
                    return false;
                }
            }
            return true;
        }

        if (s == null){
            return p == null;
        }

        if (p == null){
            return s == null;
        }

        boolean[][] result = new boolean[s.length() + 1][p.length() + 1];
        result[0][0] = true;
        for(int j = 1; j <= p.length(); j++){
            if(result[0][j - 1] && p.charAt(j - 1) == '*'){
                result[0][j] = true;
            }
        }
    }
}
```



```

    }

    for (int i = 1; i <= s.length(); i++){
        for (int j = 1; j <= p.length(); j++){
            if (p.charAt(j - 1) == s.charAt(i - 1) || p.charAt(j - 1) == '?'){
                result[i][j] = result[i - 1][j - 1];
            }
            if (p.charAt(j - 1) == '*'){
                result[i][j] = (result[i - 1][j] || result[i][j - 1]);
            }
        }
    }
    return result[s.length()][p.length()];
}
}

```

### Two Pointer version

```

public class Solution {
    /**
     * @param s: A string
     * @param p: A string includes "?" and "*"
     * @return: A boolean
     */
    public boolean isMatch(String s, String p) {
        // write your code here
        if(s.length() == 0){
            for(int i = 0; i < p.length(); i++){
                if(p.charAt(i) != '*'){
                    return false;
                }
            }
            return true;
        }

        if (s == null){
            return p == null;
        }
    }
}

```

```
    }

    if (p == null){
        return s == null;
    }

    int i = 0;
    int j = 0;
    int star = -1;
    int mark = -1;
    while(i < s.length()){
        if(j < p.length() && (p.charAt(j) == '?' || p.charAt
(j) == s.charAt(i))){
            i++;
            j++;
        }else if(j < p.length() && p.charAt(j) == '*'){
            star = j;
            j++;
            mark = i;
        }else if(star != -1){
            j = star + 1;
            i = ++mark;
        }else{
            return false;
        }
    }

    while(j < p.length() && p.charAt(j) == '*'){
        j++;
    }

    return j == p.length();
}
}
```

# Decode Ways 512

## Question

A message containing letters from A-Z is being encoded to numbers using the following mapping:

'A' -> 1

'B' -> 2

...

'Z' -> 26

Given an encoded message containing digits, determine the total number of ways to decode it.

Example

Given encoded message 12, it could be decoded as AB (1 2) or L (12).

The number of ways decoding 12 is 2.

## Solution

num[i]代表前i个元素有多少种decode方法。

初始化：num[0]=1, num[1]=s.charAt(i-1)=='0'? 0 : 1

状态函数：

分两种情况讨论

case 1: 第*i*位元素单独decode (如果为'0'则不能单独decode)

`num[i]=num[i-1]`

case 2: 第*i*位元素和前一位元素一起decode (第*i*位元素和第*i-1*位元素组成的两位数必须在10~26范围内)

`num[i]+=num[i-2]`

代码如下：

```
public class Solution {
    /**
     * @param s a string,  encoded message
     * @return an integer, the number of ways decoding
     */
    public int numDecodings(String s) {
        // Write your code here
        if(s == null || s.length() == 0){
            return 0;
        }

        int n = s.length();
        int[] num = new int[n + 1];
        num[0] = 1;
        num[1] = s.charAt(0) != '0' ? 1 : 0;

        for(int i = 2; i <= n; i++){
            if(s.charAt(i - 1) != '0'){
                num[i] = num[i - 1];
            }

            int val = (s.charAt(i - 2) - '0') * 10 + s.charAt(i
- 1) - '0';
            if(val >= 10 && val <= 26){
                num[i] += num[i - 2];
            }
        }

        return num[n];
    }
}
```

# Regular Expression Matching 154

## Question

Implement regular expression matching with support for '.' and '\*'.

'.' Matches any single character.

'\*' Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

The function prototype should be:

```
bool isMatch(const char s, const char p)
```

Some examples:

```
isMatch("aa","a") → false
```

```
isMatch("aa","aa") → true
```

```
isMatch("aaa","aa") → false
```

```
isMatch("aa","a*") → true
```

```
isMatch("aa",".*") → true
```

```
isMatch("ab",".*") → true
```

```
isMatch("aab","cab") → true
```

## Solution

这道题和[Wildcard Match](#)有相似，但是又不同。在wildcard match中，“”表示的是可以和任意字符串进行匹配，而在这道题中“”表示的是可以取任意多个其之前的字符，比如“a”就表示“”可以取任意多个“a”，例如0个（“”），1个（“a”），2个（“aa”）...。但是，这道题也可以用DP来解决，而且思想和wildcard match非常相似。

`match[i][j]`表示s中前i个字符能否和p中前j个字符匹配。

初始化：

```
match[0][0] = true
match[0][j] = (p.charAt(j - 1) == '*' && match[0][j - 2])
```

即如果p中第j位是“\*”，则对第j-1位的字符取0个（即第j-1到第j位字符看作“”），所以`match[0][j]=match[0][j-2]`

状态函数：

如果p中第j位是“.”或者p中第j位和s中第i位相同  
`match[i][j] = match[i - 1][j - 1]`

如果p中第j位是“\*”，则要分情况讨论

1. 如果p中第j-1位和s中第i位不想等并且p中第j-1位不是“.”，则不能用p中j-1位和s中第i位匹配，必须将j-1位消掉，因此“\*”取0个之前元素，`match[i][j]=match[i][j-2]`
  2. 如果p中第j-1位和s中第i位想等或者p中第j-1位为“.”，则“\*”可以有三种选择
    - 1) “\*”取0个之前元素，和1中一样，将第j-1位消掉，让j-2位去和i位匹配（“”），`match[i][j]=match[i][j-2]`
    - 2) “\*”取1个之前元素，让j-1位和i位匹配，`match[i][j]=match[i][j-1]`
    - 3) “\*”取多个之前元素，因为i位一定会被匹配掉，因此看i-1位能否继续匹配，`match[i][j]=match[i-1][j]`
- 三种情况有一种为true则`match[i][j]`为true

代码如下：

```
public class Solution {
    public boolean isMatch(String s, String p) {
        if(s == null || p == null){
            return false;
        }

        int n = s.length();
        int m = p.length();
        boolean[][] match = new boolean[n + 1][m + 1];
        match[0][0] = true;
```

```
        for(int i = 1; i <= n; i++){
            match[i][0] =false;
        }

        for(int j = 2; j <= m; j++){
            match[0][j] = (p.charAt(j - 1) == '*' && match[0][j
- 2]);
        }

        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= m; j++){
                if(p.charAt(j - 1) == '.' || p.charAt(j - 1) ==
s.charAt(i - 1)){
                    match[i][j] = match[i - 1][j - 1];
                }

                if(p.charAt(j - 1) == '*' && j > 1){
                    if(p.charAt(j - 2) != s.charAt(i - 1) && p.c
harAt(j - 2) != '.'){
                        match[i][j] = match[i][j - 2];
                    }else{
                        match[i][j] = (match[i][j - 2] || match[
i][j - 1] || match[i - 1][j]);
                    }
                }
            }
        }

        return match[n][m];
    }
}
```



# Longest Valid Parentheses 32(LeetCode)

## Question

Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.

For "()", the longest valid parentheses substring is "()", which has length = 2.

Another example is "()()()", where the longest valid parentheses substring is "()()", which has length = 4.

## Solution

这道题求最值，首先想到的是DP。

DP[i]表示以S[i]结尾的（即必须包括S[i]）的longest valid parentheses。

状态函数：

1. 如果 $S[i] == '('$ ，则以S[i]结尾的序列必然不可能是合法的，因此 $DP[i]=0$ 。
2. 如果 $S[i] == ')'$ ，则如果DP[i - 1]的最长序列前一位（假设为j）存在且为'('， $j=i-1-DP[i-1]$ ，则S[j]可以和S[i]匹配，其长度为 $DP[i-1]+2$ （即以i-1位结尾的最长字符串加上S[j]和S[i]），同时和以j-1位结尾的最长字符串连起来组成新的最长字符串。注意j和j-1位不一定存在。如果第j位不存在或者无法和S[i]匹配( $S[j] != '('$ )，则 $DP[i]=0$ 。

这里可以证明，不存在 $j' < j$ ，且 $s[j' : i]$ 为valid parentheses substring。

证明：

假如存在这样的 $j'$ ，如果则 $s[j' : i]$ 为valid parentheses substring，则 $s[j'+1 : i-1]$ 也一定为valid（前后字符匹配，剩下的中间字符必须全部valid才能使得整体都是valid），这和i-1位的最长字符串只到j+1不符合。

第二种方法受到valid parentheses的启发，可以用stack来解，只不过栈中保存的是index。

1. 遇到 '(' 无条件入栈

2. 遇到 ')' 则需要和栈中保存的 '(' 配对消除，这时候又分两种情况：

1) 栈空。则之前的 valid 串到此为止，需要将此时的位置记录为新的开始位置，即用一个 **start** 来记录此时的 **index**。

2) 栈不空。如果栈中 '(' 出栈之后栈依然不空，则将栈顶元素的 **index** 作为边界，用此时的 **index** 减去栈顶元素的 **index** 即可求出新长度。如果出栈后栈变为空，则将 **start** 记录的位置作为边界，用此时的 **index** 减去 **start** 即可求出新长度。将新长度与最大长度比较，如果更大则更新最大长度。

代码如下：

```
public class Solution {
    public int longestValidParentheses(String s) {
        if(s == null || s.length() <= 1){
            return 0;
        }

        //stack version
        // Stack<Integer> stack = new Stack<Integer>();
        // int start = -1;
        // int maxLength = 0;
        // for(int i = 0; i < s.length(); i++){
        //     if(s.charAt(i) == '('){
        //         stack.push(i);
        //     }

        //     if(s.charAt(i) == ')'){
        //         if(!stack.isEmpty()){
        //             stack.pop();
        //             if(!stack.isEmpty()){
        //                 maxLength = Math.max(maxLength, i - s
        // tack.peek());
        //             }else{
        //                 maxLength = Math.max(maxLength, i - s
        // tart);
        //             }
        //         }else{
        //             start = i;
        //         }
        //     }
        // }
```

```
//      }
//    }
// }

// return maxLength;

//DP version
char[] S = s.toCharArray();
int[] DP = new int[S.length];
int max = 0;
for(int i = 1; i < S.length; i++){
    if(S[i] == ')'){
        int j = i - 1 - DP[i - 1];
        if(j >= 0 && S[j] == '('){
            DP[i] = 2 + DP[i - 1] + (j > 0? DP[j - 1] :
0);
        }
    }

    max = Math.max(max, DP[i]);
}

return max;
}
```

# Search/Recursion/Backtracking

# Divide and Conquer

# Fast Power 140

## Question

Calculate the  $a^n \% b$  where  $a$ ,  $b$  and  $n$  are all 32bit integers.

Example

For  $2^{31} \% 3 = 2$

For  $100^{1000} \% 1000 = 0$

Challenge

$O(\log n)$

## Solution

首先%的公式为  $(a \ b) \% p = (a \% p \ b \% p) \% p$

因此这道题可以将  $a^n$  拆分：

当  $n$  为奇数时：

$a^n \% b$

$= (a^{(n/2)} a^{(n/2)} a) \% b$

$= ((a^{(n/2)} a^{(n/2)}) \% b \ a \% b) \% b$

当  $n$  为偶数时：

$a^n \% b$

$= (a^{(n/2)} * a^{(n/2)}) \% b$

代码如下：

```
class Solution {
    /*
     * @param a, b, n: 32bit integers
     * @return: An integer
     */
    public int fastPower(int a, int b, int n) {
        // write your code here
        //corner case
        if(b == 0 || n < 0){
            return -1;
        }

        if(b == 1){
            return 0;
        }

        if(n == 0 || a == 1){
            return 1 % b;
        }

        if(n == 1){
            return a % b;
        }
        //a^(1/2) % b
        long product = fastPower(a, b, n / 2);
        //((a^(1/2) % b) * (a^(1/2) % b)) % b
        product = (product * product) % b;
        //当n为奇数时，还要再乘以一个a
        if(n % 2 == 1){
            //(product % b * a % b) % b = (product * a) % b
            product = (product * a) % b;
        }

        return (int) product;
    }
};
```





# Median of two Sorted Arrays 65

## Question

There are two sorted arrays A and B of size m and n respectively. Find the median of the two sorted arrays.

Example

Given A=[1,2,3,4,5,6] and B=[2,3,4,5], the median is 3.5.

Given A=[1,2,3] and B=[4,5], the median is 3.

Challenge

The overall run time complexity should be  $O(\log(m+n))$ .

## Solution

这道题是典型的二分+DC。

1. 判断总元素数量n是奇数还是偶数。如果是奇数，则结果为第 $n/2+1$ 个元素，如果是偶数，则结果为（第 $n/2$ 元素+第 $n/2+1$ 元素） $/2$ 。
2. 假设要找第k个元素。在A和B中分别寻找各自数组的第 $k/2$ 个元素，比较两个找到的元素的大小，若A中元素小，则抛弃掉A中的 $k/2$ 个元素，反之抛弃掉B中的 $k/2$ 个元素，并继续寻找A和B中的第k（这里 $k=k-k/2$ ）个元素。
3. 几个边界条件：1）当其中一个数组元素全部被抛弃时，直接返回另一个数组中的第k个元素。2）如果 $k==1$ ，则直接返回两个数组中第一个元素较小的那一个。3）如果一个数组剩余元素不足 $k/2$ 个，则抛弃另一个数组的 $k/2$ 个元素（肯定不会将要找的第k个元素抛弃掉，因为就算不足的那个数组的元素也一起被抛弃掉，抛掉的元素还是不足k个）。

代码如下：

```
class Solution {  
    /**
```

```

    * @param A: An integer array.
    * @param B: An integer array.
    * @return: a double whose format is *.5 or *.0
    */
    public double findMedianSortedArrays(int[] A, int[] B) {
        // write your code here
        int m = A.length, n = B.length;
        int k = m + n;
        if(k % 2 == 1){
            return findKth(A, 0, B, 0, k/2 + 1);
        }else{
            return (findKth(A, 0, B, 0, k/2) + findKth(A, 0, B,
0, k/2 + 1))/2;
        }
    }

    public double findKth(int[] A, int Astart, int[] B, int Bstart, int k){
        if(Astart >= A.length){
            return (double) B[Bstart + k - 1];
        }
        if(Bstart >= B.length){
            return (double) A[Astart + k - 1];
        }

        if(k == 1){
            return Math.min(A[Astart], B[Bstart]);
        }

        int A_key = Astart + k/2 - 1 < A.length? A[Astart + k/2
- 1] : Integer.MAX_VALUE;
        int B_key = Bstart + k/2 - 1 < B.length? B[Bstart + k/2
- 1] : Integer.MAX_VALUE;

        if(A_key < B_key){
            return findKth(A, Astart + k/2, B, Bstart, k - k/2);
        }else{
            return findKth(A, Astart, B, Bstart + k/2, k - k/2);
        }
    }
}

```

```
}
```

# Maximum Depth of Binary Tree 97

## Question

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example

Given a binary tree as follow:

1 /\ 2 3 /\ 4 5

The maximum depth is 3.

## Solution

寻找左右子树的max depth，选较大的那个再+1即可。

代码如下：

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: An integer.
     */
    public int maxDepth(TreeNode root) {
        // write your code here
        if(root == null){
            return 0;
        }

        int left = maxDepth(root.left);
        int right = maxDepth(root.right);

        return Math.max(left, right) + 1;
    }
}
```

# Balanced Binary Tree 93

## Question

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

Example

Given binary tree A = {3,9,20,#,#,15,7}, B = {3,#,20,15,7}

A) 3 B) 3 / \ \ 9 20 20 / \ \ 15 7 15 7

The binary tree A is a height-balanced binary tree, but B is not.

## Solution

看到binary tree就想用二分和DC。。。咳咳。。。控制一下自己。。。

很简单，首先判断左右子树是否是balanced，然后判断左右子树的max depth是不是相差1以内。寻找左右子树的max depth又是一次DC的过程，即寻找当前子树的左右子树的max depth选较大的那个再+1即可。

代码如下：

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
```

```
public class Solution {  
    /**  
     * @param root: The root of binary tree.  
     * @return: True if this Binary tree is Balanced, or false.  
     */  
    public boolean isBalanced(TreeNode root) {  
        // write your code here  
        if(root == null){  
            return true;  
        }  
  
        if(!isBalanced(root.left) || !isBalanced(root.right)){  
            return false;  
        }  
  
        if(Math.abs(maxDepth(root.left)-maxDepth(root.right)) >  
1){  
            return false;  
        }  
  
        return true;  
    }  
  
    public int maxDepth(TreeNode root){  
        if(root == null){  
            return 0;  
        }  
  
        int left = maxDepth(root.left);  
        int right = maxDepth(root.right);  
  
        return Math.max(left, right) + 1;  
    }  
}
```

# Pow(x, n) 428

## Question

Implement pow(x, n).

Notice

You don't need to care about the precision of your answer, it's acceptable if the expected answer and your answer 's difference is smaller than  $1e-3$ .

Example

$\text{Pow}(2.1, 3) = 9.261$

$\text{Pow}(0, 1) = 0$

$\text{Pow}(1, 0) = 1$

Challenge

$O(\log n)$  time

## Solution

三种解法，都非常straight forward。

其中D & C是二分法，所以时间复杂度是 $O(\lg n)$ 。

注意：leetcode中的test case里考虑n为MIN\_VALUE(-2147483648)，这时候如果直接取反会造成溢出(MAX\_VALUE=2147483647)，因此要先将n+1变成-2147483647再取反才行。同时，因为我们将偶数次幂变成了奇数次幂，如果 $x < 0$ ，我们要将x取反。

代码如下：

recursion version



```
public class Solution {
    /**
     * @param x the base number
     * @param n the power number
     * @return the result
     */
    public double myPow(double x, int n) {
        // Write your code here
        if(n == 0){
            return 1;
        }

        if(x == 0){
            return 0;
        }

        if(n > 0){
            return power(x, n);
        }else{
            return 1 / power(x, -n);
        }
    }

    private double power(double x, int n){
        if(n == 1){
            return x;
        }

        double v = power(x, n / 2);
        if(n % 2 == 0){
            return v * v;
        }else{
            return v * v * x;
        }
    }
}
```

D & C version

```
public class Solution {
    /**
     * @param x the base number
     * @param n the power number
     * @return the result
     */
    public double myPow(double x, int n) {
        // Write your code here
        if(n == 0){
            return 1;
        }

        if(n == 1){
            return x;
        }

        if(x == 0){
            return 0;
        }

        boolean isNegative = false;
        if(n < 0){
            isNegative = true;
            n *= -1;
        }

        int k = n / 2;
        int l = n - k * 2;
        double v1 = myPow(x, k);
        double v2 = myPow(x, l);
        if(isNegative){
            return 1 / (v1 * v1 * v2);
        }else{
            return v1 * v1 * v2;
        }
    }
}
```

Non-recursion (for loop) version

```
public class Solution {  
    /**  
     * @param x the base number  
     * @param n the power number  
     * @return the result  
     */  
    public double myPow(double x, int n) {  
        // Write your code here  
        // double prod = 1.0;  
        // if(x == 0){  
        //     if(n == 0){  
        //         return 1;  
        //     }else{  
        //         return 0;  
        //     }  
        // }else if(n == 0){  
        //     return 1;  
        // }else{  
        //     if(n > 0){  
        //         for(int i = 0; i < n; i++){  
        //             prod = prod * x;  
        //         }  
        //     }else{  
        //         for(int i = 0; i < -n; i++){  
        //             prod = prod / x;  
        //         }  
        //     }  
        //     return prod;  
        // }  
    }  
}
```

# Find Minimum in Rotated Sorted Array II 160

## Question

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

Find the minimum element.

Notice

The array may contain duplicates.

Example

Given [4,4,5,6,7,0,1,2] return 0.

## Solution

这道题和I不一样，因为有重复元素，所以不能用二分查找的方法（如1，1，-1，1就会有问题），只能遍历所有元素求最小值。

代码如下：

```
public class Solution {  
    /**  
     * @param num: a rotated sorted array  
     * @return: the minimum number in the array  
     */  
    public int findMin(int[] num) {  
        // write your code here  
        if (num == null || num.length == 0){  
            return -1;  
        }  
        int mini = 0;  
  
        for(int i = 0; i < num.length; i++){  
            if(num[i] < num[mini]){  
                mini = i;  
            }  
        }  
  
        return num[mini];  
    }  
}
```

# Merge k Sorted Lists 104

## Question

Merge k sorted linked lists and return it as one sorted list.

Analyze and describe its complexity.

Example

Given lists:

[ 2->4->null, null, -1->null ],

return -1->2->4->null.

## Solution

这道题可以用3种方法解。

第一种方法：D&C。总共有n条sorted list，将前 $n/2$ 条进行合并，将后 $n/2$ 条进行合并，再将左右进行合并。

第二种方法：merge 2 by 2。和第一种方法类似，两条两条merge，比如1-2，3-4，。。。，将merge完的加入到新的lists中。如果n为奇数，最后一条没法和下一条merge，直接加入到新lists中。一直这样两两merge直到只剩下1条为止。

第三种方法：用PriorityQueue(heap)实现。将每个list的开头node加入pq中，根据node的value大小从小到大排序。然后将队头元素（最小元素）出队，并将该元素的next元素入队。一直重复，直到队列为空为止。

代码如下：

```
/**
 * Definition for ListNode.
 * public class ListNode {
 *     int val;
 *     ListNode next;

```

```
*     ListNode(int val) {
*         this.val = val;
*         this.next = null;
*     }
* }
*/
public class Solution {
    /**
     * @param lists: a list of ListNode
     * @return: The head of one sorted list.
     */
    //version1: D&C
    /*
    private ListNode merge(ListNode left, ListNode right){
        ListNode dummy = new ListNode(0);
        ListNode head = dummy;

        while(left != null && right != null){
            if(left.val < right.val){
                head.next = left;
                left = left.next;
            }else{
                head.next = right;
                right = right.next;
            }
            head = head.next;
        }

        if(left != null){
            head.next = left;
        }

        if(right != null){
            head.next = right;
        }

        return dummy.next;
    }

    public ListNode mergeKLists(List<ListNode> lists) {
```

```
// write your code here
if(lists == null || lists.size() == 0){
    return null;
}

if(lists.size() == 1){
    return lists.get(0);
}

int size = lists.size();
List<ListNode> leftLists = new ArrayList<ListNode>();
List<ListNode> rightLists = new ArrayList<ListNode>();

for(int i = 0; i < size; i++){
    if(i < size/2){
        leftLists.add(lists.get(i));
    }else{
        rightLists.add(lists.get(i));
    }
}

ListNode left = mergeKLists(leftLists);
ListNode right = mergeKLists(rightLists);

return merge(left, right);
}*/

/*
//version2: merge 2 by 2
public ListNode mergeKLists(List<ListNode> lists) {
    // write your code here
    if(lists == null || lists.size() == 0){
        return null;
    }

    while(lists.size() > 1){
        List<ListNode> newLists = new ArrayList<ListNode>();
        for(int i = 0; i < lists.size() - 1; i++){
            if(i % 2 == 0){
```



```
        ListNode node = merge(lists.get(i), lists.get(i + 1));
        newLists.add(node);
    }
}
if(lists.size() % 2 != 0){
    newLists.add(lists.get(lists.size() - 1));
}
lists = newLists;
}

return lists.get(0);
}

private ListNode merge(ListNode left, ListNode right){
    ListNode dummy = new ListNode(0);
    ListNode head = dummy;

    while(left != null && right != null){
        if(left.val < right.val){
            head.next = left;
            left = left.next;
        }else{
            head.next = right;
            right = right.next;
        }

        head = head.next;
    }

    if(left != null){
        head.next = left;
    }

    if(right != null){
        head.next = right;
    }

    return dummy.next;
}
```

```
*/
//ListNode[] heapArray;
//heapsort version
private Comparator<ListNode> ListNodeComparator = new Compar
ator<ListNode>() {
    public int compare(ListNode left, ListNode right) {
        // if (left == null) {
        //     return 1;
        // } else if (right == null) {
        //     return -1;
        // }
        return left.val - right.val;
    }
};

public ListNode mergeKLists(List<ListNode> lists) {
    if (lists == null || lists.size() == 0) {
        return null;
    }

    Queue<ListNode> heap = new PriorityQueue<ListNode>(lists
.size(), ListNodeComparator);
    for (int i = 0; i < lists.size(); i++) {
        if (lists.get(i) != null) {
            heap.add(lists.get(i));
        }
    }

    ListNode dummy = new ListNode(0);
    ListNode tail = dummy;
    while (!heap.isEmpty()) {
        ListNode head = heap.poll();
        tail.next = head;
        tail = head;
        if (head.next != null) {
            heap.add(head.next);
        }
    }
}
```

```
        return dummy.next;
    }
}
```

**DFS**

# Nested List Weight Sum 551

## Question

Given a nested list of integers, return the sum of all integers in the list weighted by their depth. Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Example

Given the list `[[1,1],2,[1,1]]`, return 10. (four 1's at depth 2, one 2 at depth 1,  $4 \times 1 + 2 \times 1 = 10$ ) Given the list `[1,[4,[6]]]`, return 27. (one 1 at depth 1, one 4 at depth 2, and one 6 at depth 3;  $1 + 4 \times 2 + 6 \times 3 = 27$ )

## Solution

Recursion: 遍历list，当前元素如果是数，则将其乘以当前深度并加到sum里，若当前元素是list，则递归遍历该list，并将深度加1。

DFS: 用一个stack来实现。遍历list，将当前元素和深度加入stack中。依次取出栈顶元素，若取出元素为数，则将其乘以其深度并加入sum，若取出元素是list，则将该list中元素依次加入stack中，同时这些元素的深度比之前取出元素深度+1。

代码如下：

```
/**
 * // This is the interface that allows for creating nested list
 * S.
 * // You should not implement it, or speculate about its implementation
 * public interface NestedInteger {
 *
 *     // @return true if this NestedInteger holds a single integer,
 *     // rather than a nested list.
 *     public boolean isInteger();
 * }
```

```

*      // @return the single integer that this NestedInteger holds,
ds,
*      // if it holds a single integer
*      // Return null if this NestedInteger holds a nested list
*      public Integer getInteger();
*
*      // @return the nested list that this NestedInteger holds,
*      // if it holds a nested list
*      // Return null if this NestedInteger holds a single integer
er
*      public List<NestedInteger> getList();
*  }
*/
class Node{
    int depth;
    NestedInteger nestedInteger;
    public Node(int depth, NestedInteger nestedInteger){
        this.depth = depth;
        this.nestedInteger = nestedInteger;
    }
}

public class Solution {
    //Recursion version
    // public int depthSum(List<NestedInteger> nestedList) {
    //     // Write your code here
    //     return helper(nestedList, 1);
    // }

    // private int helper(List<NestedInteger> nestedList, int depth){
    //     int n = nestedList.size();
    //     int sum = 0;
    //     for(int i = 0; i < n; i++){
    //         if(nestedList.get(i).isInteger()){
    //             sum += depth * nestedList.get(i).getInteger();
    //         }else{
    //             sum += helper(nestedList.get(i).getList(), depth + 1);
    //         }
    //     }
    //     return sum;
    // }
}

```

```
//      }
//      }
//      return sum;
// }

//DFS version
public int depthSum(List<NestedInteger> nestedList) {
    // Write your code here
    Stack<Node> stack = new Stack<Node>();
    for(int i = 0; i < nestedList.size(); i++){
        stack.push(new Node(1, nestedList.get(i)));
    }

    int sum = 0;
    while(!stack.isEmpty()){
        Node curt = stack.pop();
        if(curt.nestedInteger.isInteger()){
            sum += curt.depth * curt.nestedInteger.getInteger();
        }else{
            List<NestedInteger> list = curt.nestedInteger.getList();
            for(int i = 0; i < list.size(); i++){
                stack.push(new Node(curt.depth + 1, list.get(i)));
            }
        }
    }

    return sum;
}
```

# Route Between Two Nodes in Graph 176

## Question

Given a directed graph, design an algorithm to find out whether there is a route between two nodes.

Example

Given graph:

A----->B----->C \ | | | | \ v ->D----->E

for s = B and t = E, return true

for s = D and t = C, return false

## Solution

典型的DFS求解。用stack来实现dfs，用set来记录元素是否曾经被加入过。

代码如下：

```
/**
 * Definition for Directed graph.
 * class DirectedGraphNode {
 *     int label;
 *     ArrayList<DirectedGraphNode> neighbors;
 *     DirectedGraphNode(int x) {
 *         label = x;
 *         neighbors = new ArrayList<DirectedGraphNode>();
 *     }
 * };
 */
public class Solution {
    /**
     * @param graph: A list of Directed graph node
     * @param s: the starting Directed graph node
```



```
* @param t: the terminal Directed graph node
* @return: a boolean value
*/
public boolean hasRoute(ArrayList<DirectedGraphNode> graph,
                        DirectedGraphNode s, DirectedGraphNode
de t) {
    // write your code here
    if(graph == null || s == null || t == null){
        return false;
    }

    Stack<DirectedGraphNode> stack = new Stack<DirectedGraph
Node>();
    HashSet<DirectedGraphNode> set = new HashSet<DirectedGra
phNode>();
    stack.push(s);
    set.add(s);

    while(!stack.isEmpty()){
        DirectedGraphNode curt = stack.pop();
        if(curt == t){
            return true;
        }
        for(DirectedGraphNode neighbor : curt.neighbors){
            if(!set.contains(neighbor)){
                stack.push(neighbor);
                set.add(neighbor);
            }
        }
    }

    return false;
}
}
```

# Recursion

# Print Numbers by Recursion 371

## Question

Print numbers from 1 to the largest number with N digits by recursion.

Notice

It's pretty easy to do recursion like:

```
recursion(i) { if i > largest number: return results.add(i) recursion(i + 1) }
```

however this cost a lot of recursion memory as the recursion depth maybe very large. Can you do it in another way to recursive with at most N depth?

Example

Given N = 1, return [1,2,3,4,5,6,7,8,9].

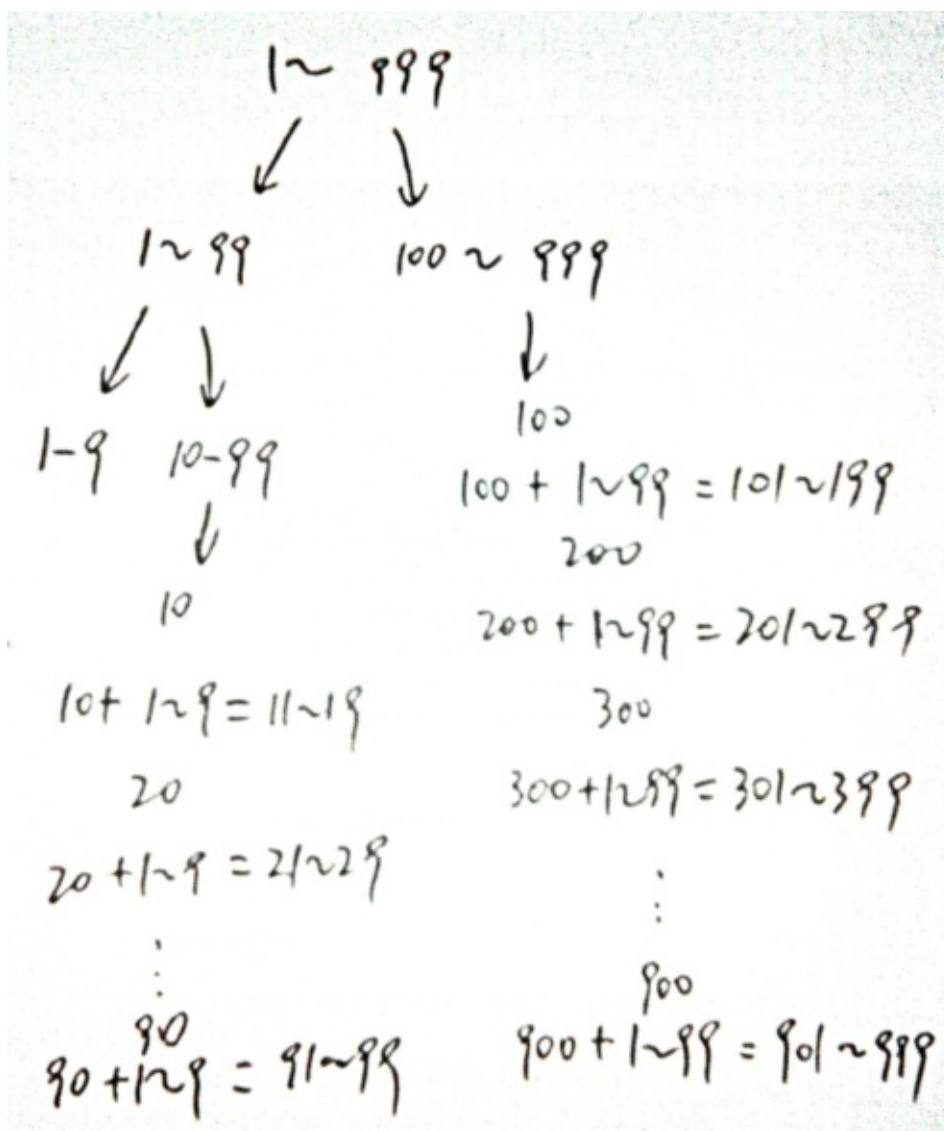
Given N = 2, return [1,2,3,4,5,6,7,8,9,10,11,12,...,99].

Challenge

Do it in recursion, not for-loop.

## Solution

在求n-1位数到n位数的时候，先求n-2位数到n-1位数，就如同下面一样：



递归求解 $n$ 时，可以对 $n-1$ 位置上的数从0-9遍历，依次类推直到最后一位。

代码如下：

```
public class Solution {
    /**
     * @param n: An integer.
     * return : An array storing 1 to the largest number with n
    digits.
     */
    public List<Integer> numbersByRecursion(int n) {
        // write your code here
        List<Integer> res = new ArrayList<Integer>();
        if(n < 1){
            return res;
        }

        helper(n, 0, res);
        return res;
    }

    //ans在这里可以看成前一位的值，在递归过程中会乘以n-1次10，即10^(n-1)
    private void helper(int n, int ans, List<Integer> res){
        if(n==0){
            if(ans>0){
                res.add(ans);
            }
            return;
        }

        int i;
        for(i=0; i<=9; i++){
            helper(n-1, ans*10+i, res);
        }
    }
}
```

# Restore IP Addresses 426

## Question

Given a string containing only digits, restore it by returning all possible valid IP address combinations.

Example

Given "25525511135", return

[ "255.255.11.135",

"255.255.111.35" ]

Order does not matter.

## Solution

这道题其实是将字符串分割成4段，使得结果为合法的IP地址。要求返回所有合法IP地址，想到用递归。

合法的IP地址为：1) 由4段组成 2) 每一段长度为1到3 3) 每一段如果不是0，不能以0开头，如01，00 4) 每一段大小不能大于255

1. 用一个result记录合法答案，用一个list纪录当前答案，start代表能够取值的起始位置。
2. 当前段可以取start~start，start~start+1，start~start+2这几个字符串，然后递归求下一段。每次取值要检验取的字符串的合法性。
3. 当取满4段时，看此时是否还有字符串剩下，没有则将4段拼成合法的IP地址

代码如下：

```
public class Solution {  
    /**  
     * @param s the IP string  
     * @return All possible valid IP addresses
```

```

    */
    public ArrayList<String> restoreIpAddresses(String s) {
        // Write your code here
        ArrayList<String> result = new ArrayList<String>();
        if(s == null || s.length() == 0){
            return result;
        }

        ArrayList<String> list = new ArrayList<String>();
        helper(s, result, list, 0);
        return result;
    }

    private void helper(String s, ArrayList<String> result, ArrayList<String> list, int start){
        if(list.size() == 4){
            //分成4段后若还有字符串剩余则不合法
            if(start != s.length()){
                return;
            }
            //每一段后面加".", 再将最后一个"."删去
            StringBuilder sb = new StringBuilder();
            for(String str : list){
                sb.append(str);
                sb.append(".");
            }
            sb.deleteCharAt(sb.length() - 1);
            result.add(sb.toString());
            return;
        }

        for(int i = 1; i <= 3 && start + i <= s.length(); i++){
            //每一段长度不能超过3, 同时如果超过了1位, 则不能以0开头, 如01
            if(i != 1 && s.charAt(start) == '0'){
                break;
            }
            //如果该段为3位, 则要检验是否超过255
            if(i == 3 && !isValid(s.substring(start, start + 3)))
        ){
            break;
        }
    }

```

```
        }

        String temp = s.substring(start, start + i);
        list.add(temp);
        helper(s, result, list, start + i);
        list.remove(list.size() - 1);
    }
}

//合法IP的每一段都小于等于255
private boolean isValid(String s){

    if(Integer.valueOf(s) > 255){
        return false;
    }
    return true;
}
}
```



# Generate Parentheses 427

## Question

Given  $n$  pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

Example

Given  $n = 3$ , a solution set is:

"((()))", "(()())", "(())()", "()(())", "()()()"

## Solution

要求所有组合，用递归。

用temp记录当前的括号组合，用left记录左括号数量，用right记录右括号数量。

1. 如果在某次递归时，左括号的个数大于右括号的个数，说明此时生成的字符串中右括号的个数大于左括号的个数，即会出现')('这样的非法串，所以这种情况直接返回，不继续处理。
2. 如果left和right都为0，则说明此时生成的字符串已有3个左括号和3个右括号，且字符串合法，则存入结果中后返回。
3. 如果以上两种情况都不满足，若此时left大于0，则调用递归函数，注意参数的更新（左括号数量减少1，temp+"("），若right大于0，则调用递归函数，同样要更新参数（右括号数量减少1,temp+")"）。
4. 如果左括号数量已经为0而右括号数量不为0，可以直接将右括号补全而不调用递归

代码如下：

```
public class Solution {  
    /**  
     * @param n n pairs
```

```
* @return All combinations of well-formed parentheses
*/
public ArrayList<String> generateParenthesis(int n) {
    // Write your code here
    ArrayList<String> res = new ArrayList<String>();
    if(n < 1){
        return res;
    }
    helper(n, n, "", res);
    return res;
}

private void helper(int left, int right, String temp, ArrayList<String> res){
    if(left < 0 || right < 0 || left > right){
        return;
    }

    if(left == 0 && right == 0){
        res.add(temp);
        return;
    }

    if(left == 0){
        while(right != 0){
            temp += ")";
            right--;
        }
        res.add(temp);
        return;
    }

    helper(left - 1, right, temp + "(", res);
    helper(left, right - 1, temp + ")", res);
}
}
```

# Letter Combinations of a Phone Number 425

## Question

Given a digit string excluded 01, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.



Notice

Although the above answer is in lexicographical order, your answer could be in any order you want.

Example

Given "23"

Return ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"]

## Solution

求所有组合自然想到递归。

首先用一个hashmap将每个数字及其对应的字母加入。然后递归求digits上每一位数字对应字母的所有组合。

代码如下：

```
public class Solution {
```

```
/**
 * @param digits A digital string
 * @return all posible letter combinations
 */
public ArrayList<String> letterCombinations(String digits) {
    // Write your code here
    ArrayList<String> res = new ArrayList<String>();
    if(digits == null || digits.length() == 0){
        return res;
    }

    HashMap<Character, char[]> map = new HashMap<Character,
char[]>();
    map.put('0', new char[] {});
    map.put('1', new char[] {});
    map.put('2', new char[] { 'a', 'b', 'c' });
    map.put('3', new char[] { 'd', 'e', 'f' });
    map.put('4', new char[] { 'g', 'h', 'i' });
    map.put('5', new char[] { 'j', 'k', 'l' });
    map.put('6', new char[] { 'm', 'n', 'o' });
    map.put('7', new char[] { 'p', 'q', 'r', 's' });
    map.put('8', new char[] { 't', 'u', 'v' });
    map.put('9', new char[] { 'w', 'x', 'y', 'z' });

    StringBuilder sb = new StringBuilder();
    helper(digits, map, res, sb, 0);
    return res;
}

private void helper(String digits, HashMap<Character, char[]
> map, ArrayList<String> res, StringBuilder sb, int pos){
    if(sb.length() == digits.length()){
        res.add(sb.toString());
        return;
    }

    //sb.length()表明当前在哪一位上
    for(char c : map.get(digits.charAt(pos))){
        sb.append(c);
        helper(digits, map, res, sb, pos + 1);
    }
}
```

```
        sb.deleteCharAt(sb.length() - 1);  
    }  
}  
}
```

# Backtracking

# Word Search 123

## Question

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example

Given board =

[ "ABCE",

"SFCS",

"ADEE" ]

word = "ABCCED", -> returns true,

word = "SEE", -> returns true,

word = "ABCB", -> returns false.

## Solution

对matrix进行搜索，当第一个字符匹配上之后，再看下一个字符。每次对当前位置的上下左右四个方向进行搜索，有一个满足即返回true。在对当前位置的邻接位置搜索之前，用"#"覆盖当前位置的值，这样在新的搜索时不会取到之前取过的重复位置，避免重复。当四个方向均不满足时，将当前位置恢复为原来值，然后对当前位置进行新的搜索。

代码如下：

```
public class Solution {  
    /**  
     * @param board: A list of lists of character
```

```
* @param word: A string
* @return: A boolean
*/
public boolean exist(char[][] board, String word) {
    // write your code here
    if(board == null || board.length == 0){
        return false;
    }

    if(word == null || word.length() == 0){
        return true;
    }

    for(int i = 0; i < board.length; i++){
        for(int j = 0; j < board[0].length; j++){
            if(board[i][j] == word.charAt(0)){
                boolean res = findWord(i, j, board, word, 0)
;

                if(res){
                    return true;
                }
            }
        }
    }

    return false;
}

private boolean findWord(int i, int j, char[][] board, String word, int start){
    //start代表新搜索开始的位置，当和word长度相等时，表示word中所有
    字符均已匹配
    if(start == word.length()){
        return true;
    }

    //考虑边界
    if(i < 0 || j < 0 || i >= board.length || j >= board[0].
length || board[i][j] != word.charAt(start)){
```



```
        return false;
    }

    //mark当前位置
    board[i][j] = '#';

    //对当前位置的上下左右四个方向搜索，有一个满足即可
    if(findWord(i - 1, j, board, word, start + 1) || findWord(i + 1, j, board, word, start + 1) || findWord(i, j - 1, board, word, start + 1) || findWord(i, j + 1, board, word, start + 1))
    {
        return true;
    }

    //将当前位置恢复，以进行下一次新的搜索
    board[i][j] = word.charAt(start);

    return false;
}
}
```

## Followup Question

Word Search II

[https://www.gitbook.com/book/zhengyang2015/lintcode/edit#/edit/master/word\\_search\\_ii\\_132.md](https://www.gitbook.com/book/zhengyang2015/lintcode/edit#/edit/master/word_search_ii_132.md)

# Word Ladder II 121

## Question

Given two words (start and end), and a dictionary, find all shortest transformation sequence(s) from start to end, such that:

Only one letter can be changed at a time

Each intermediate word must exist in the dictionary

Notice

All words have the same length.

All words contain only lowercase alphabetic characters.

Example

Given:

start = "hit"

end = "cog"

dict = ["hot","dot","dog","lot","log"]

Return

[ ["hit","hot","dot","dog","cog"],

```
["hit","hot","lot","log","cog"]
```

]

## Solution

求最短距离的所有解，利用BFS+DFS。

1) BFS过程与Word Ladder类似，但是在BFS过程中利用一个HashMap(distance)记录每个单词距start的最短距离，另一个HashMap(map)记录每个单词是由哪些单词变换一步得到的。

2) DFS过程类似Permutation，从end出发，逐个寻找其上一节点直到start。其中可能的上一节点可以由1)中的map得到，但是只有距离比当前节点距离小1的上一节点才是合法的（保证其一定在end和start之间的最短路径上，即找到的下一个点一定比当前点离start更近，这样防止搜到的下一个点向外走），距离可以由1)中的distance得到。当到达start时即找到一种解，因为是从end出发，所以要reverse成正序后再加入res，然后reverse回逆序后删除刚加入元素，再回溯寻找下一个解。

注意：在dfs获取next的list的时候，需要判断curr是否在map中，否则可能得到null而造成之后的运行时异常。比如start="hot"，end="dog"，dict={"hot", "dog"}，这里在bfs中就找不到hot和dog的next，因此没有元素会被加到map和distance中。在lintcode里可以过，但是在leetcode里过不了。

代码如下：

```
public class Solution {
    /**
     * @param start, a string
     * @param end, a string
     * @param dict, a set of string
     * @return a list of lists of string
     */
    public List<List<String>> findLadders(String start, String end, Set<String> dict) {
        // write your code here
        List<List<String>> result = new ArrayList<List<String>>();

        if(dict == null){
            return result;
        }

        dict.add(start);
        dict.add(end);

        HashMap<String, ArrayList<String>> map = new HashMap<String, ArrayList<String>>();
        HashMap<String, Integer> distance = new HashMap<String,
```

```

Integer>());
    Queue<String> queue = new LinkedList<String>();
    queue.offer(start);
    distance.put(start, 0);

    bfs(map, distance, queue, dict);

    List<String> list = new ArrayList<String>();

    dfs(result, list, end, start, distance, map);

    return result;
}

private void dfs(List<List<String>> result, List<String> list, String curt, String start, HashMap<String, Integer> distance, HashMap<String, ArrayList<String>> map){
    list.add(curt);
    if(curt.equals(start)){
        Collections.reverse(list);
        result.add(new ArrayList<String>(list));
        Collections.reverse(list);
    }else{
        if(map.containsKey(curt)){
            for(String next : map.get(curt)){
                if(distance.containsKey(next) && distance.get(curt) == distance.get(next) + 1){
                    dfs(result, list, next, start, distance, map);
                }
            }
        }
        list.remove(list.size() - 1);
    }
}

private void bfs(HashMap<String, ArrayList<String>> map, HashMap<String, Integer> distance, Queue<String> queue, Set<String> dict){
    while(!queue.isEmpty()){

```

```

        int size = queue.size();
        for(int i = 0; i < size; i++){
            String curt = queue.poll();
            for(String next : expend(curt, dict)){
                if(!map.containsKey(next)){
                    map.put(next, new ArrayList<String>());
                    map.get(next).add(curt);
                }else{
                    map.get(next).add(curt);
                }

                if(!distance.containsKey(next)){
                    distance.put(next, distance.get(curt) +
1);

                    queue.offer(next);
                }
            }
        }
    }

    private ArrayList<String> expend(String word, Set<String> dict){
        ArrayList<String> nextWords = new ArrayList<String>();
        for(char c = 'a'; c <= 'z'; c++){
            for(int i = 0; i < word.length(); i++){
                if(c == word.charAt(i)){
                    continue;
                }
                String newWord = replace(word, i, c);
                if(dict.contains(newWord)){
                    nextWords.add(newWord);
                }
            }
        }
        return nextWords;
    }

    private String replace(String word, int index, char c){
        char[] charac = word.toCharArray();

```

```
        charac[index] = c;  
        return new String(charac);  
    }  
}
```

# Combination Sum 135

## Question

Given a set of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

The same repeated number may be chosen from C unlimited number of times.

For example, given candidate set 2,3,6,7 and target 7,

A solution set is:

[7]

[2, 2, 3]

Notice

All numbers (including target) will be positive integers.

Elements in a combination (a1, a2, ... , ak) must be in non-descending order. (ie,  $a_1 \leq a_2 \leq \dots \leq a_k$ ).

The solution set must not contain duplicate combinations.

## Solution

与Subsets类似。注意几点：

- 1) 每加入一个数，就要把下一层的target为原来的target减去这个数
- 2) 需要考虑避免重复：对元素排序，判断是否和前一个数等值
- 3) 可以重复取相同元素，所以下一层的起始pos和上一层相同

代码如下：

```
public class Solution {  
    /**
```

```
* @param candidates: A list of integers
* @param target: An integer
* @return: A list of lists of integers
*/
public ArrayList<ArrayList<Integer>> combinationSum(int[] candidates, int target) {
    // write your code here
    ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
    if(candidates == null || candidates.length == 0 || target <= 0){
        return result;
    }

    ArrayList<Integer> list = new ArrayList<Integer>();
    Arrays.sort(candidates);
    helper(result, list, candidates, target, 0);

    return result;
}

private void helper(ArrayList<ArrayList<Integer>> result, ArrayList<Integer> list, int[] candidates, int target, int pos){
    if(target == 0){
        result.add(new ArrayList<Integer>(list));
        return;
    }

    for(int i = pos; i < candidates.length; i++){
        if(i != 0 && candidates[i] == candidates[i - 1]){
            continue;
        }

        if(candidates[i] > target){
            return;
        }
        list.add(candidates[i]);
        helper(result, list, candidates, target - candidates[i], i);
        list.remove(list.size() - 1);
    }
}
```



```
    }  
  }  
}
```

# Combination Sum II 153

## Question

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

Each number in C may only be used once in the combination.

Notice

All numbers (including target) will be positive integers.

Elements in a combination ( $a_1, a_2, \dots, a_k$ ) must be in non-descending order. (ie,  $a_1 \leq a_2 \leq \dots \leq a_k$ ).

The solution set must not contain duplicate combinations.

Example

Given candidate set [10,1,6,7,2,1,5] and target 8,

A solution set is:

[ [1,7],

[1,2,5],

[2,6],

[1,1,6] ]

## Solution

与Combination Sum类似。但是因为每个数只能用一次，下一层pos为上一层pos+1。避免重复的方法：重复的数只能取第一个元素。

代码如下：

需要额外O(n)空间记录元素状态

```
public class Solution {
    /**
     * @param num: Given the candidate numbers
     * @param target: Given the target number
     * @return: All the combinations that sum to target
     */
    public ArrayList<ArrayList<Integer>> combinationSum2(int[] num, int target) {
        // write your code here
        ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
        if(num == null || num.length == 0 || target <= 0){
            return result;
        }

        ArrayList<Integer> list = new ArrayList<Integer>();
        Arrays.sort(num);
        int[] visit = new int[num.length];
        helper(result, list, num, target, 0, visit);

        return result;
    }

    private void helper(ArrayList<ArrayList<Integer>> result, ArrayList<Integer> list, int[] num, int target, int pos, int[] visit){
        if(target == 0){
            result.add(new ArrayList<Integer>(list));
            return;
        }

        for(int i = pos; i < num.length; i++){
            if(visit[i] == 1 || (i != 0 && num[i] == num[i - 1] && visit[i - 1] == 0)){
                continue;
            }

            if(num[i] > target){
                return;
            }
        }
    }
}
```

```

        list.add(num[i]);
        visit[i] = 1;
        helper(result, list, num, target - num[i], i + 1, visit);
        list.remove(list.size() - 1);
        visit[i] = 0;
    }
}

```

Revised version:

```

public class Solution {

    private ArrayList<ArrayList<Integer>> results;

    public ArrayList<ArrayList<Integer>> combinationSum2(int[] candidates,
        int target) {
        if (candidates.length < 1) {
            return results;
        }

        ArrayList<Integer> path = new ArrayList<Integer>();
        java.util.Arrays.sort(candidates);
        results = new ArrayList<ArrayList<Integer>> ();
        combinationSumHelper(path, candidates, target, 0);

        return results;
    }

    private void combinationSumHelper(ArrayList<Integer> path, int[] candidates, int sum, int pos) {
        if (sum == 0) {
            results.add(new ArrayList<Integer>(path));
        }

        if (pos >= candidates.length || sum < 0) {
            return;
        }
    }
}

```

```
    }

    int prev = -1;
    for (int i = pos; i < candidates.length; i++) {
        //重复的数只能取第一个元素
        if (candidates[i] != prev) {
            path.add(candidates[i]);
            combinationSumHelper(path, candidates, sum - candidates[i], i + 1);
            prev = candidates[i];
            path.remove(path.size()-1);
        }
    }
}

}
```

## Combinations 152

### Question

Given two integers  $n$  and  $k$ , return all possible combinations of  $k$  numbers out of  $1 \dots n$ .

Example

For example,

If  $n = 4$  and  $k = 2$ , a solution is:

`[[2,4],[3,4],[2,3],[1,2],[1,3],[1,4]]`

### Solution

Comination模版。注意是从1开始到 $n$ 。

代码如下：

```
public class Solution {
    /**
     * @param n: Given the range of numbers
     * @param k: Given the numbers of combinations
     * @return: All the combinations of k numbers out of 1..n
     */
    public ArrayList<ArrayList<Integer>> combine(int n, int k) {
        // write your code here
        ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
        if(n <= 0 || k <= 0 || n < k){
            return result;
        }

        ArrayList<Integer> list = new ArrayList<Integer>();
        combineHelper(result, list, n, k, 1);

        return result;
    }

    private void combineHelper(ArrayList<ArrayList<Integer>> result, ArrayList<Integer> list, int n, int k, int pos){
        if(list.size() == k){
            result.add(new ArrayList<Integer>(list));
            return;
        }

        for(int i = pos; i <= n; i++){
            list.add(i);
            combineHelper(result, list, n, k, i + 1);
            list.remove(list.size() - 1);
        }
    }
}
```

# Binary Search



# Search in Rotated Sorted Array 62

## Question

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

Example

For [4, 5, 1, 2, 3] and target=1, return 2.

For [4, 5, 1, 2, 3] and target=0, return -1.

Challenge

$O(\log N)$  time

## Solution

如果没有重复元素，就是二分搜索，mid要么落在大元素的区间，要么落在小元素的区间。先看mid落在前半区间还是后半区间，再看target是在该区间的前半部分还是后半部分。

代码如下：

No duplicate

```
public class Solution {  
    /**  
     * @param A : an integer rotated sorted array  
     * @param target : an integer to be searched  
     * @return : an integer  
     */  
}
```

```
public int search(int[] A, int target) {
    // write your code here
    if (A == null || A.length == 0){
        return -1;
    }

    int start = 0;
    int end = A.length - 1;
    while (start + 1 < end){
        int mid = start + (end - start)/2;
        if (A[mid] == target){
            return mid;
        }
        if (A[start] < A[mid]){
            if (A[start] <= target && target <= A[mid]){
                end = mid;
            }else{
                start = mid;
            }
        }else{
            if (A[mid] <= target && target <= A[end]){
                start = mid;
            }else{
                end = mid;
            }
        }
    }

    if (A[start] == target){
        return start;
    }else if (A[end] == target){
        return end;
    }else{
        return -1;
    }
}
```

With duplicate

```
public class Solution {  
    /**  
     * param A : an integer rotated sorted array and duplicates  
are allowed  
     * param target : an integer to be search  
     * return : a boolean  
     */  
    public boolean search(int[] A, int target) {  
        // write your code here  
        if(A == null || A.length == 0){  
            return false;  
        }  
  
        for(int i = 0; i < A.length; i++){  
            if(A[i] == target){  
                return true;  
            }  
        }  
  
        return false;  
    }  
}
```

# Search in Rotated Sorted Array II 63

## Question

Follow up for Search in Rotated Sorted Array:

What if duplicates are allowed?

Would this affect the run-time complexity? How and why?

Write a function to determine if a given target is in the array.

Example

Given [1, 1, 0, 1, 1, 1] and target = 0, return true.

Given [1, 1, 1, 1, 1, 1] and target = 0, return false.

## Solution

这道题是有重复元素的情况下对rotated sorted array进行搜索。

一种做法是进行顺序搜索，这样复杂度是 $O(n)$ 。

另一种做法是进行二分搜索，但是因为重复元素（特别是重复元素有可能被分在前后区间），此时对于 $nums[mid] == nums[left]$ 情况不能像之前那样合并在 $nums[left] < nums[mid]$ 或者 $nums[left] > nums[mid]$ 之中进行讨论，必须进行单独讨论，因为 $nums[mid] == nums[left]$ 有可能是因为 $mid == left$ ，但也有可能是因为 $nums[mid]$ 和 $nums[left]$ 是重复元素，如果是后面这种情况则 $mid$ 有可能其实是落在后半区，这时能确定的只是 $nums[left]$ 一定不等于target，其他情况都是不确定的，因此最保险的方法是只将 $left$ 向右移动一步，即 $left++$ 。这种情况下，如果数组大部分是重复元素，则复杂度接近 $O(n)$ ，但是如果重复元素较少，则时间复杂度接近 $O(\log n)$ 。

代码如下：

```
public class Solution {  
    /**
```

```

    * param A : an integer rotated sorted array and duplicates
are allowed
    * param target : an integer to be search
    * return : a boolean
    */
public boolean search(int[] A, int target) {
    // write your code here
    int[] nums = A;
    if(nums == null || nums.length == 0){
        return false;
    }

    int left = 0;
    int right = nums.length - 1;
    while(left + 1 < right){
        int mid = left + (right - left) / 2;
        if(nums[mid] == target){
            return true;
        }else if(nums[left] < nums[mid]){
            if(nums[left] <= target && target <= nums[mid]){
                right = mid;
            }else{
                left = mid;
            }
        }else if(nums[left] > nums[mid]){
            if(nums[mid] <= target && target <= nums[right])
{
                left = mid;
            }else{
                right = mid;
            }
        }else if(nums[left] == nums[mid]){
            //这里是关键，如果有重复元素，只将left向right移动一位，肯定
不会有问题
            left++;
        }
    }

    if(nums[left] == target){
        return true;
    }
}

```

```
        }else if(nums[right] == target){
            return true;
        }
        return false;
    }
}
```

# Search for a Range 61

## Question

Given a sorted array of  $n$  integers, find the starting and ending position of a given target value.

If the target is not found in the array, return  $[-1, -1]$ .

Example

Given  $[5, 7, 7, 8, 8, 10]$  and target value 8,

return  $[3, 4]$ .

Challenge

$O(\log n)$  time.

## Solution

做两遍binary search。第一遍找第一个出现的target，第二遍找最后出现的target。

代码如下：

```
public class Solution {  
    /**  
     * @param A : an integer sorted array  
     * @param target : an integer to be inserted  
     * return : a list of length 2, [index1, index2]  
     */  
    public int[] searchRange(int[] A, int target) {  
        // write your code here  
        if (A == null || A.length == 0){  
            return new int[]{-1, -1};  
        }  
  
        int start = 0;
```

```
int end = A.length - 1;
int[] bound = new int[2];

while (start + 1 < end){
    int mid = start + (end - start)/2;
    if (A[mid] == target){
        end = mid;
    }else if (A[mid] < target){
        start = mid;
    }else{
        end = mid;
    }
}

if (A[start] == target){
    bound[0] = start;
}else if (A[end] == target){
    bound[0] = end;
}else{
    bound[0]=bound[1]=-1;
    return bound;
}

start = 0; end = A.length - 1;

while (start + 1 < end){
    int mid = start + (end - start)/2;
    if (A[mid] == target){
        start = mid;
    }else if (A[mid] < target){
        start = mid;
    }else{
        end = mid;
    }
}

if (A[end] == target){
    bound[1] = end;
}else if (A[start] == target){
    bound[1] = start;
```



```
        }else{
            bound[0]=bound[1]=-1;
            return bound;
        }
        return bound;
    }
}
```

# Search Insert Position 60

## Question

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume NO duplicates in the array.

Example

[1,3,5,6], 5 → 2

[1,3,5,6], 2 → 1

[1,3,5,6], 7 → 4

[1,3,5,6], 0 → 0

Challenge

$O(\log(n))$  time

## Solution

二分搜索，找到第一个大于等于target的数。

代码如下：

```
public class Solution {
    /**
     * param A : an integer sorted array
     * param target : an integer to be inserted
     * return : an integer
     */
    public int searchInsert(int[] A, int target) {
        // write your code here
        if(A == null || A.length == 0){
            return 0;
        }

        int start = 0;
        int end = A.length - 1;

        while (start + 1 < end){
            int mid = start + (end - start)/2;
            if (A[mid] == target){
                return mid;
            }else if (A[mid] < target){
                start = mid;
            }else{
                end = mid;
            }
        }

        if (A[start] >= target){
            return start;
        }else if (A[end] < target){
            return end + 1;
        }else{
            return end;
        }
    }
}
```

# BFS

# Word Ladder 120

## Question

Given two words (start and end), and a dictionary, find the length of shortest transformation sequence from start to end, such that:

Only one letter can be changed at a time

Each intermediate word must exist in the dictionary

Notice

Return 0 if there is no such transformation sequence.

All words have the same length.

All words contain only lowercase alphabetic characters.

Example

Given:

start = "hit"

end = "cog"

dict = ["hot","dot","dog","lot","log"]

As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",

return its length 5.

## Solution

在每一步只能变换一个字符的情况下，根据字典中的变幻规则，求两个字符串之间的最短距离。最短距离用BFS。

1) 将start和end都加入字典中。

- 2) 对于每一个字符串，所有能经过一步变幻得到的在字典中的字符串为其下一层元素。
- 3) 出现过的元素存在一个HashSet中，防止元素往上一层走。
- 4) 以start为起始点，用BFS逐层寻找end。在每一层上，将该层上每一个元素的所有下一层元素加入队列（不重复）。
- 5) 直到某一元素的下一层元素中包含end，返回当前length。

Follow up: [Word Ladder II](#)

代码如下：

```
public class Solution {  
    /**  
     * @param start, a string  
     * @param end, a string  
     * @param dict, a set of string  
     * @return an integer  
     */  
    public int ladderLength(String start, String end, Set<String>  
> dict) {  
        // write your code here  
        if(dict == null){  
            return 0;  
        }  
  
        if(start.equals(end)){  
            return 1;  
        }  
  
        HashSet<String> hash = new HashSet<String>();  
        Queue<String> queue = new LinkedList<String>();  
        dict.add(end);  
        queue.offer(start);  
        hash.add(start);  
  
        //BFS  
        int length = 1;  
        while(!queue.isEmpty()){
```

```
        length++;
        int size = queue.size();
        for(int i = 0; i < size; i++){
            String word = queue.poll();
            for(String next : getNextWord(word, dict)){
                if(next.equals(end)){
                    return length;
                }

                if(!hash.contains(next)){
                    queue.offer(next);
                    hash.add(next);
                }
            }
        }
    }

    return 0;
}

private String replace(String word, int index, char c){
    char[] characters = word.toCharArray();
    characters[index] = c;
    return new String(characters);
}

private ArrayList<String> getNextWord(String word, Set<String> dict){
    ArrayList<String> list = new ArrayList<String>();
    for(char i = 'a'; i <= 'z'; i++){
        for(int j = 0; j < word.length(); j++){
            if(i == word.charAt(j)){
                continue;
            }
            String newWord = replace(word, j, i);
            if(dict.contains(newWord)){
                list.add(newWord);
            }
        }
    }
}
```













```
        return list;
    }
}
```





# 数据结构

1) Queue 2) Stack 3) Hash 4) Heap

8 - Data Structure		Required (4/10)	Optional (1/13)
Easy	128. Hash Function		16 %
Medium	544. Top k Largest Numbers <i>new</i>	 	23 %
Medium	486. Merge k Sorted Arrays		20 %
Medium	129. Rehashing		25 %
Medium	104. Merge k Sorted Lists	 	26 %
Medium	40. Implement Queue by Two Stacks		38 %
Medium	12. Min Stack	 	30 %
Medium	4. Ugly Number II		21 %
Hard	134. LRU Cache	 	19 %
Hard	122. Largest Rectangle in Histogram		24 % <a href="http://www.voidcn.com">www.voidcn.com</a>
8 - Data Structure		Required (4/10)	Optional (1/13)
Easy	495. Implement Stack	 	37 %
Easy	494. Implement Stack by Two Queues		24 %
Easy	493. Implement Queue by Linked List II		27 %
Easy	492. Implement Queue by Linked List		30 %
Easy	128. Hash Function		16 %
Medium	545. Top k Largest Numbers II <i>new</i>	 	26 %
Medium	229. Stack Sorting		30 %
Medium	471. Top K Frequent Words		15 %
Medium	130. Heapify		33 %
Medium	124. Longest Consecutive Sequence		31 %
Hard	230. Animal Shelter		22 %
Hard	24. LFU Cache		17 %
Hard	126. Max Tree		29 % <a href="http://www.voidcn.com">www.voidcn.com</a>

# Tree

# Heap

Filter	Question No.	Heap	Pick one for me
Hard	81. Data Stream Median		26 %
Medium	104. Merge k Sorted Lists		27 %
Medium	130. Heapify		34 %
Super	131. Building Outline		12 %
Medium	401. Kth Smallest Number in Sorted Matrix		19 %
Medium	518. Super Ugly Number		25 %

sift-up  $O(n \log n)$

sift-down  $O(n)$

build heap  $O(n)$

具体可以查算法导论堆排序。

index从0开始

已知子节点坐标为 $i$ ，父节点坐标： $(i-1) / 2$

已知父节点坐标为 $i$ ，左子节点坐标为 $2i+1$ ，右子节点坐标为 $2i+2$

# Top k largest numbers II 545

## Question

Implement a data structure, provide two interfaces:

1. `add(number)`. Add a new number in the data structure.
2. `topk()`. Return the top k largest numbers in this data structure. k is given when we create the data structure.

Example

```
s = new Solution(3);
```

create a new data structure.

```
s.add(3)
```

```
s.add(10)
```

```
s.topk()
```

```
return [10, 3]
```

```
s.add(1000)
```

```
s.add(-99)
```

```
s.topk()
```

```
return [1000, 10, 3]
```

```
s.add(4)
```

```
s.topk()
```

```
return [1000, 10, 4]
```

```
s.add(100)
```

```
s.topk()
```

return [1000, 100, 10]

## Solution

用一个PriorityQueue(MinHeap)来保存前k大个数：

- 1) 新来数时，当PriorityQueue小于k个数时，直接加入
- 2) 当PriorityQueue大于等于k个数时，若新来的数比堆顶的数(堆中最小值，即第k大的数)大时，删去堆顶数后再加入新来的数

代码如下：

```
class topKLargests{
    PriorityQueue<Integer> queue;
    int k;

    public topKLargests(int k){
        queue = new PriorityQueue<Integer>();
        this.k = k;
    }

    public void add(int number){
        if(queue.size() < k){
            queue.offer(number);
        }else{
            if(queue.peek() < number){
                queue.poll();
                queue.offer(number);
            }
        }
    }

    public void topk(){
        Stack<Integer> stack = new Stack<Integer>();
        PriorityQueue<Integer> queue1 = new PriorityQueue(queue)
;

        while(!queue1.isEmpty()){
            stack.push(queue1.poll());
        }
    }
}
```

```
    }

    ArrayList<Integer> res = new ArrayList<Integer>();

    while(!stack.isEmpty()){
        res.add(stack.pop());
    }

    for(int i : res){
        System.out.print(i + " ");
    }
    System.out.println();
}
}
```

### Reversed topk()

```
public List<Integer> topk() {
    Iterator iter = minheap.iterator();
    List<Integer> result = new ArrayList<Integer>();
    while (iter.hasNext()) {
        result.add((Integer) iter.next());
    }
    Collections.sort(result, Collections.reverseOrder());
    return result;
}
```

# Kth Smallest Number in Sorted Matrix 401

## Question

Find the kth smallest number in at row and column sorted matrix.

Example

Given  $k = 4$  and a matrix:

[ [1 ,5 ,7],

[3 ,7 ,8],

[4 ,8 ,9], ]

return 5

## Solution

用一个PriorityQueue(Heap)来实现。PriorityQueue堆顶保存的为当前元素中最小的元素：

1) 先将 $k$ 个第一列元素加入PriorityQueue（若第一列元素个数小于 $k$ ，加入全部第一列元素）。此时，堆顶元素为 $matrix[0][0]$ ，为当前最小元素， $matrix[1][0]$ 为第二小元素。

2) 将堆顶元素出队，加入其右边元素 $matrix[0][1]$ 。若 $matrix[1][0]$ 比 $matrix[0][1]$ 小，则 $matrix[1][0]$ 为堆顶元素，且为当前元素中最小元素（比它下面，右边， $matrix[0][1]$ 及其右边元素都小）；反之，则 $matrix[0][1]$ 为堆顶元素，且为当前元素中最小元素。因此，将出队元素（当前最小值）的右边元素入队已定能保证该性质。

3) 重复2)中过程，每次出队的都是当前元素中最小元素，出队 $k-1$ 此之后的堆顶元素即为第 $k$ 个元素。

代码如下：

```
public class Solution {
```



```
/**
 * @param matrix: a matrix of integers
 * @param k: an integer
 * @return: the kth smallest number in the matrix
 */

public class Point {
    public int x, y, val;
    public Point(int x, int y, int val) {
        this.x = x;
        this.y = y;
        this.val = val;
    }
}

Comparator<Point> comp = new Comparator<Point>() {
    public int compare(Point left, Point right) {
        return left.val - right.val;
    }
};

public int kthSmallest(int[][] matrix, int k) {
    if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {
        return 0;
    }
    if (k > matrix.length * matrix[0].length) {
        return 0;
    }
    return horizontal(matrix, k);
}

private int horizontal(int[][] matrix, int k) {
    Queue<Point> heap = new PriorityQueue<Point>(k, comp);
    for (int i = 0; i < Math.min(matrix.length, k); i++) {
        heap.offer(new Point(i, 0, matrix[i][0]));
    }
    for (int i = 0; i < k - 1; i++) {
        Point curr = heap.poll();
        if (curr.y + 1 < matrix[0].length) {

```

```
        heap.offer(new Point(curr.x, curr.y + 1, matrix[
curr.x][curr.y + 1]));
    }
}
return heap.peek().val;
}
}
```

# Super Ugly Number 518

## Question

Write a program to find the  $n$ th super ugly number.

Super ugly numbers are positive numbers whose all prime factors are in the given prime list primes of size  $k$ . For example,  $[1, 2, 4, 7, 8, 13, 14, 16, 19, 26, 28, 32]$  is the sequence of the first 12 super ugly numbers given primes =  $[2, 7, 13, 19]$  of size 4.

Notice

1 is a super ugly number for any given primes.

The given numbers in primes are in ascending order.

$0 < k \leq 100$ ,  $0 < n \leq 10^6$ ,  $0 < \text{primes}[i] < 1000$

Example

Given  $n = 6$ , primes =  $[2, 7, 13, 19]$  return 13

## Solution

用HashMap + PriorityQueue。取 $n-1$ 次最小堆堆顶元素，最后堆顶元素为第 $n$ 个prime。

代码如下：

```
public class Solution {
    /**
     * @param n a positive integer
     * @param primes the given prime list
     * @return the nth super ugly number
     */
    public int nthSuperUglyNumber(int n, int[] primes) {
        // Write your code here
        if(primes == null || primes.length == 0 || n <= 0){
            return 0;
        }

        Queue<Long> Q = new PriorityQueue<Long>();
        HashSet<Long> inQ = new HashSet<Long>();
        Q.add(Long.valueOf(1));
        inQ.add(Long.valueOf(1));

        for(int i = 1; i < n; i++){
            Long curt = Q.poll();
            for(int j = 0; j < primes.length; j++){
                Long next = curt * primes[j];
                if(!inQ.contains(next)){
                    Q.add(next);
                    inQ.add(next);
                }
            }
        }

        return Q.peek().intValue();
    }
}
```

## Related Question

[Ugly Number II 4](#)



# Kth Largest in N Arrays 543

## Question

Find K-th largest element in N arrays.

Notice

You can swap elements in the array

Example

In n=2 arrays [[9,3,2,4,7],[1,2,3,4,8]], the 3rd largest element is 7.

In n=2 arrays [[9,3,2,4,8],[1,2,3,4,2]], the 1st largest element is 9, 2nd largest element is 8, 3rd largest element is 7 and etc.

## Solution

1. 将每一行从小到大排序
2. 将每一行最大（最后）的元素加入一个最大堆
3. 取出堆顶元素，若此时取出元素数量不足k个，则将取出元素的左边元素（如果存在）加入堆
4. 重复3直到取出k个元素为止

代码如下：

```
public class Solution {  
    /**  
     * @param arrays a list of array  
     * @param k an integer  
     * @return an integer, K-th largest element in N arrays  
     */  
    class Node{  
        int value;  
        int row;
```

```
int column;
public Node(int row, int column, int value){
    this.value = value;
    this.row = row;
    this.column = column;
}
}

public int KthInArrays(int[][] arrays, int k) {
    // Write your code here
    if(arrays == null || arrays.length == 0 || k <= 0){
        return -1;
    }

    for(int i = 0; i < arrays.length; i++){
        Arrays.sort(arrays[i]);
    }

    Queue<Node> queue = new PriorityQueue<Node>(arrays.length, new Comparator<Node>(){
        public int compare(Node a, Node b){
            return b.value - a.value;
        }
    });

    for(int i = 0; i < arrays.length; i++){
        if(arrays[i].length > 0){
            queue.offer(new Node(i, arrays[i].length - 1, arrays[i][arrays[i].length - 1]));
        }
    }

    for(int i = 0; i < k; i++){
        Node temp = queue.poll();
        if(i == k - 1){
            return temp.value;
        }
        if(temp.column > 0){
            queue.offer(new Node(temp.row, temp.column - 1, arrays[temp.row][temp.column - 1]));
        }
    }
}
```

```
        }  
    }  
  
    return -1;  
}  
}
```



# Kth Smallest Sum In Two Sorted Arrays

## 465

### Question

Given two integer arrays sorted in ascending order and an integer  $k$ . Define  $\text{sum} = a + b$ , where  $a$  is an element from the first array and  $b$  is an element from the second one. Find the  $k$ th smallest sum out of all possible sums.

Example

Given  $[1, 7, 11]$  and  $[2, 4, 6]$ .

For  $k = 3$ , return 7.

For  $k = 4$ , return 9.

For  $k = 8$ , return 15.

Challenge

Do it in either of the following time complexity:

$O(k \log \min(n, m, k))$ . where  $n$  is the size of  $A$ , and  $m$  is the size of  $B$ .

$O((m + n) \log \text{maxValue})$ . where  $\text{maxValue}$  is the max number in  $A$  and  $B$ .

### Solution

idea：位置为  $i$  ( $A$ 中)， $j$  ( $B$ 中) 元素 (表示  $A[i] + B[j]$ ) 的下一个元素为位置  $i+1, j$  和  $i, j+1$  之中的一个。

1. 将  $A[0]$  和  $B[0]$  的和加入一个最小堆，并将位置  $0, 0$  加入一个 `hashset` 记录加入过堆的位置
2. 将最堆顶元素出队，得到其位置  $i, j$ ，并将  $A[i+1] + B[j]$  和  $A[i] + B[j+1]$  加入堆中，其中  $i+1$  和  $j+1$  必须是合法的，即不越界，同时  $i+1, j$  和  $i, j+1$  没有加入过堆
3. 重复2，出队  $k$  次即可

代码如下：

```
public class Solution {
    /**
     * @param A an integer arrays sorted in ascending order
     * @param B an integer arrays sorted in ascending order
     * @param k an integer
     * @return an integer
     */
    class Node implements Comparable<Node>{
        int i;
        int j;
        int val;
        public Node(int i, int j, int val){
            this.i = i;
            this.j = j;
            this.val = val;
        }

        @Override
        public int compareTo(Node another){
            if(this.val == another.val){
                return 0;
            }
            return this.val > another.val? 1 : -1;
        }
    }

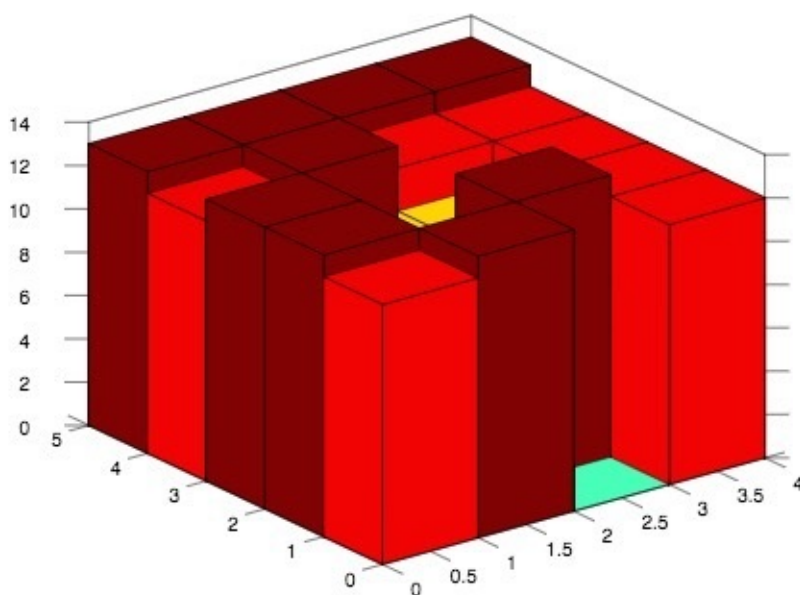
    int[] di = new int[]{1, 0};
    int[] dj = new int[]{0, 1};
    public int kthSmallestSum(int[] A, int[] B, int k) {
        // Write your code here
        if(A == null || B == null || A.length == 0 && B.length == 0 || k > A.length * B.length){
            return 0;
        }else if(A.length == 0){
            return B[k];
        }else if(B.length == 0){
            return A[k];
        }
    }
}
```

```
HashSet<String> set = new HashSet<String>();
PriorityQueue<Node> queue = new PriorityQueue<Node>();
Node temp = new Node(0, 0, A[0] + B[0]);
queue.offer(temp);
set.add(temp.i + "," + temp.j);
Node next;
int i = 0;
int j = 0;
for(int n = 0; n < k - 1; n++){
    temp = queue.poll();
    for(int m = 0; m < 2; m++){
        i = temp.i + di[m];
        j = temp.j + dj[m];
        next = new Node(i, j, 0);
        if(i >= 0 && i < A.length && j >= 0 && j < B.length && !set.contains(next.i + "," + next.j)){
            next.val = A[i] + B[j];
            queue.offer(next);
            set.add(next.i + "," + next.j);
        }
    }
}
return queue.peek().val;
}
```

# Trapping Rain Water II 364

## Question

Given  $n \times m$  non-negative integers representing an elevation map 2d where the area of each cell is  $1 \times 1$ , compute how much water it is able to trap after raining.



Example

Given  $5 \times 4$  matrix

[12,13,0,12]

[13,4,13,12]

[13,8,10,12]

[12,13,12,12]

[13,13,13,13]

return 14.

## Solution

这道题和Trapping Rain Water类似，也是由边缘最低处开始逐渐向内部灌水。用一个pq来保存当前边缘元素。

tips：用dx，dy来做相邻搜索。

1. 将上下左右四条边加入pq，用一个2维数组来记录每个点是否为边缘。
2. 取pq堆顶元素（最小元素），探索该元素上下左右四个方向元素（在数组范围内且不是边缘），若其相邻元素比当前元素小将其差值计入总数，然后将相邻元素加入pq，并标记为边缘；若其相邻元素比当前元素大，则直接加入pq，并标记为边缘。
3. 直到pq中没有元素为止

代码如下：

```
public class Solution {
    /**
     * @param heights: a matrix of integers
     * @return: an integer
     */
    class Node{
        int x;
        int y;
        int height;
        public Node(int x, int y, int height){
            this.x = x;
            this.y = y;
            this.height = height;
        }
    }

    int[] dx = {-1, 1, 0, 0};
    int[] dy = {0, 0, -1, 1};
    public int trapRainWater(int[][] heights) {
        // write your code here
        if(heights == null || heights.length <= 2 || heights[0].length <= 2){
            return 0;
        }
    }
}
```

```
int n = heights.length;
int m = heights[0].length;
boolean[][] isVisited = new boolean[n][m];

Queue<Node> queue = new PriorityQueue<Node>(1, new Comp
rator<Node>(){
    public int compare(Node a, Node b){
        return a.height - b.height;
    }
});
//add first and last column
for(int i = 0; i < n; i++){
    queue.offer(new Node(i, 0, heights[i][0]));
    isVisited[i][0] = true;
    queue.offer(new Node(i, m - 1, heights[i][m - 1]));
    isVisited[i][m - 1] = true;
}
//add first and last row
for(int j = 1; j < m - 1; j++){
    queue.offer(new Node(0, j, heights[0][j]));
    isVisited[0][j] = true;
    queue.offer(new Node(n - 1, j, heights[n - 1][j]));
    isVisited[n - 1][j] = true;
}

int sum = 0;
while(!queue.isEmpty()){
    Node curt = queue.poll();
    for(int i = 0; i < 4; i++){
        int nextX = curt.x + dx[i];
        int nextY = curt.y + dy[i];
        if(nextX >= 0 && nextX < n && nextY >= 0 && next
Y < m && !isVisited[nextX][nextY]){
            if(heights[nextX][nextY] < curt.height){
                sum += curt.height - heights[nextX][next
Y];
                queue.offer(new Node(nextX, nextY, curt.
height));
                isVisited[nextX][nextY] = true;
            }else{
```

```
        queue.offer(new Node(nextX, nextY, heights[nextX][nextY]));
        isVisited[nextX][nextY] = true;
    }
}
}
return sum;
};
```

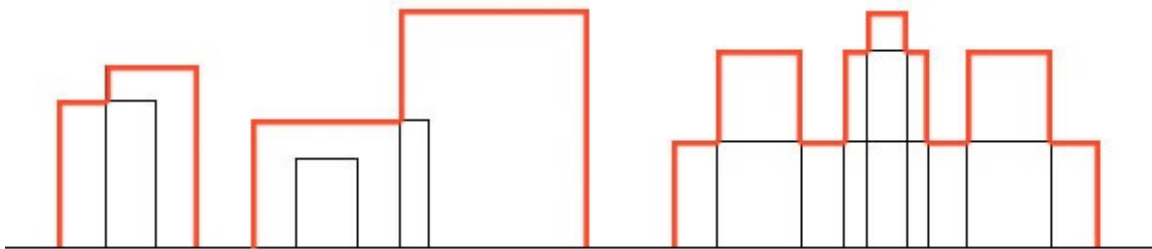
# Building Outline 131

## Question

Given N buildings in a x-axis , each building is a rectangle and can be represented by a triple (start, end, height) , where start is the start position on x-axis, end is the end position on x-axis and height is the height of the building.

Buildings may overlap if you see them from far away , find the outline of them .

An outline can be represented by a triple, (start, end, height), where start is the start position on x-axis of the outline, end is the end position on x-axis and height is the height of the outline.



Notice

Please merge the adjacent outlines if they have the same height and make sure different outlines cant overlap on x-axis.

Example

Given 3 buildings :

[ [1, 3, 3],

[2, 4, 4],

[5, 6, 1] ]

The outlines are :

[ [1, 2, 3],

[2, 4, 4],



[5, 6, 1]

## Solution

我的解法：用一个带最大堆的扫描线遍历数组，每出现一个拐点则记录一次区间。新加入一个元素后若堆顶元素和之前不同，则表明出现拐点。

1. 首先处理数组中元素。将每一个点用一个point来保存，保存time（开始写错了，应该是位置，但是要改的太多了），flag（起点为1，终点为0），height。用一个HashMap来记录每一对终点和起点（终点为key，起点为value）。
2. 将所有point保存在一个list中并排序，首先根据时间从小到大排序，若时间相等则根据flag排序（先起点后终点），若flag也相等则根据height排序（若同为起点则从大到小排序，若同为终点则从小到大排序）。这样可以避免重复解。
3. 再构建一个最大堆，用于记录加入的元素的最大值。
4. 开始遍历list中每个point，起点元素加入堆，终点元素删去堆中对应的起点元素。
5. 当遇到一个起点元素时，先记录加入前堆顶元素，然后将该起点元素加入，再看加入后堆顶元素，1）若没有变化，则继续下一个point；2）若有变化，则说明出现拐点，将之前堆顶元素时间作为起点，当前堆顶元素时间作为终点，之前堆顶元素高度作为高度。注意：就算堆顶元素变化，但是如果之前堆顶元素和当前堆顶元素时间相同，说明是在这个时间连续有几个起点被加入，宽度为0，不能算一个区间。
6. 当遇到一个终点元素时，将其对应的起点元素从堆中删除。若此时堆为空，则和5中一样记录一个区间，并继续下一个point。若堆不为空，则需要看此时堆顶元素是否改变。若不变则继续，否则说明出现“拐点”。此处“拐点”要分两种情况讨论：1）若新的堆顶元素高度和之前堆顶元素高度相同，则说明相同高度的两段区间有重叠，题目要求若发生这种情况要合并这两段区间，所以我们要保留之前的堆顶元素（两段同高度相同重叠区间的最左边），删去新的堆顶元素（即代替原堆顶元素被删除，因为每遇到一个终点必须删去一个起点），具体做法可以先删去新堆顶元素，再加入原堆顶元素，或者直接将新堆顶元素时间改为原堆顶元素时间。2）若新堆顶和原堆顶元素高度不同，则像5中那样记录一个区间，但是要将现在的堆顶元素时间改为遇到的终点元素的时间。

## 7. 遍历完整个list结束

九章解法：用HashMap解，有hashmap模版。

代码如下：

```
public class Solution {
    /**
     * @param buildings: A list of lists of integers
     * @return: Find the outline of those buildings
     */
    class Point{
        int time;
        //1 -> start, 0 -> end
        int flag;
        int height;
        public Point(int time, int flag, int height){
            this.time = time;
            this.flag = flag;
            this.height = height;
        }
    }

    public ArrayList<ArrayList<Integer>> buildingOutline(int[][]
buildings) {
        // write your code here
        ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayL
ist<Integer>>();
        if(buildings == null || buildings.length == 0 || buildin
gs[0].length == 0){
            return res;
        }

        HashMap<Point, Point> map = new HashMap<Point, Point>();
        ArrayList<Point> list = new ArrayList<Point>();
        for(int i = 0; i < buildings.length; i++){
            Point start = new Point(buildings[i][0], 1, building
s[i][2]);
            Point end = new Point(buildings[i][1], 0, buildings[
i][2]);
```

```
        list.add(start);
        list.add(end);
        map.put(end, start);
    }
    Collections.sort(list, new Comparator<Point>(){
        public int compare(Point a, Point b){
            if(a.time == b.time){
                if(a.flag == b.flag){
                    if(a.flag == 1){
                        return b.height - a.height;
                    }else{
                        return a.height - b.height;
                    }
                }else{
                    return b.flag - a.flag;
                }
            }else{
                return a.time - b.time;
            }
        }
    });
});
```

```
    Queue<Point> queue = new PriorityQueue<Point>(1, new Com
parator<Point>(){
        public int compare(Point a, Point b){
            if(a.height == b.height){
                return a.time - b.time;
            }else{
                return b.height - a.height;
            }
        }
    });
});
```

```
for(int i = 0; i < list.size(); i++){
    if(queue.size() == 0){
        queue.offer(list.get(i));
        continue;
    }
}
```

```
    ArrayList<Integer> temp = new ArrayList<Integer>();
```

```
Point pre = queue.peek();
Point next = list.get(i);
if(next.flag == 1){
    queue.offer(next);
    Point curt = queue.peek();
    if(curt != pre){
        if(curt.time == pre.time){
            continue;
        }
        temp.add(pre.time);
        temp.add(curt.time);
        temp.add(pre.height);
        res.add(temp);
    }else{
        continue;
    }
}else{
    queue.remove(map.get(next));
    if(queue.size() == 0){
        temp.add(pre.time);
        temp.add(next.time);
        temp.add(pre.height);
        res.add(temp);
        continue;
    }

    Point curt = queue.peek();
    if(curt != pre){
        if(curt.height == pre.height){
            curt.time = pre.time;
            continue;
        }
        temp.add(pre.time);
        temp.add(next.time);
        temp.add(pre.height);
        res.add(temp);
        curt.time = next.time;
    }else{
        continue;
    }
}
```

```
        }
    }

    return res;
}
}
```

HashMap:

```
public class Solution {
    //HashMap模版
    class HashHeap {
        ArrayList<Integer> heap;
        String mode;
        int size_t;
        HashMap<Integer, Node> hash;

        class Node {
            public Integer id;
            public Integer num;

            Node(Node now) {
                id = now.id;
                num = now.num;
            }

            Node(Integer first, Integer second) {

                this.id = first;
                this.num = second;
            }
        }

        public HashHeap(String mod) {
            // TODO Auto-generated constructor stub
            heap = new ArrayList<Integer>();
            mode = mod;
            hash = new HashMap<Integer, Node>();
            size_t = 0;
        }
    }
}
```

```
}

public int peek() {
    return heap.get(0);
}

public int size() {
    return size_t;
}

public Boolean isEmpty() {
    return (heap.size() == 0);
}

int parent(int id) {
    if (id == 0) {
        return -1;
    }
    return (id - 1) / 2;
}

int lson(int id) {
    return id * 2 + 1;
}

int rson(int id) {
    return id * 2 + 2;
}

boolean comparesmall(int a, int b) {
    if (a <= b) {
        if (mode == "min")
            return true;
        else
            return false;
    } else {
        if (mode == "min")
            return false;
        else
            return true;
    }
}
```

```
    }

}

void swap(int idA, int idB) {
    int valA = heap.get(idA);
    int valB = heap.get(idB);

    int numA = hash.get(valA).num;
    int numB = hash.get(valB).num;
    hash.put(valB, new Node(idA, numB));
    hash.put(valA, new Node(idB, numA));
    heap.set(idA, valB);
    heap.set(idB, valA);
}

public Integer poll() {
    size_t--;
    Integer now = heap.get(0);
    Node hashnow = hash.get(now);
    if (hashnow.num == 1) {
        swap(0, heap.size() - 1);
        hash.remove(now);
        heap.remove(heap.size() - 1);
        if (heap.size() > 0) {
            siftdown(0);
        }
    } else {
        hash.put(now, new Node(0, hashnow.num - 1));
    }
    return now;
}

public void add(int now) {
    size_t++;
    if (hash.containsKey(now)) {
        Node hashnow = hash.get(now);
        hash.put(now, new Node(hashnow.id, hashnow.num + 1));
    } else {
```

```
        heap.add(now);
        hash.put(now, new Node(heap.size() - 1, 1));
    }

    siftup(heap.size() - 1);
}

public void delete(int now) {
    size_t--;
    Node hashnow = hash.get(now);
    int id = hashnow.id;
    int num = hashnow.num;
    if (hashnow.num == 1) {

        swap(id, heap.size() - 1);
        hash.remove(now);
        heap.remove(heap.size() - 1);
        if (heap.size() > id) {
            siftup(id);
            siftdown(id);
        }
    } else {
        hash.put(now, new Node(id, num - 1));
    }
}

void siftup(int id) {
    while (parent(id) > -1) {
        int parentId = parent(id);
        if (comparesmall(heap.get(parentId), heap.get(id)) == true) {
            break;
        } else {
            swap(id, parentId);
        }
        id = parentId;
    }
}

void siftdown(int id) {
```



```
while (lson(id) < heap.size()) {
    int leftId = lson(id);
    int rightId = rson(id);
    int son;
    if (rightId >= heap.size()
        || (comparesmall(heap.get(leftId), heap.get(rightId)
) == true)) {
        son = leftId;
    } else {
        son = rightId;
    }
    if (comparesmall(heap.get(id), heap.get(son)) == true) {
        break;
    } else {
        swap(id, son);
    }
    id = son;
}
}
```

```
class Edge {
    int pos;
    int height;
    boolean isStart;

    public Edge(int pos, int height, boolean isStart) {
        this.pos = pos;
        this.height = height;
        this.isStart = isStart;
    }
}
```

```
class EdgeComparator implements Comparator<Edge> {
    @Override
    public int compare(Edge arg1, Edge arg2) {
        Edge l1 = (Edge) arg1;
        Edge l2 = (Edge) arg2;
        if (l1.pos != l2.pos)
```

```

        return compareInteger(l1.pos, l2.pos);
    if (l1.isStart && l2.isStart) {
        return compareInteger(l2.height, l1.height);
    }
    if (!l1.isStart && !l2.isStart) {
        return compareInteger(l1.height, l2.height);
    }
    return l1.isStart ? -1 : 1;
}

```

```

int compareInteger(int a, int b) {
    return a <= b ? -1 : 1;
}
}

```

```

ArrayList<ArrayList<Integer>> output(ArrayList<ArrayList<Integer>> res) {

```

```

    ArrayList<ArrayList<Integer>> ans = new ArrayList<ArrayList<Integer>>();

```

```

    if (res.size() > 0) {

```

```

        int pre = res.get(0).get(0);

```

```

        int height = res.get(0).get(1);

```

```

        for (int i = 1; i < res.size(); i++) {

```

```

            ArrayList<Integer> now = new ArrayList<Integer>();

```

```

            int id = res.get(i).get(0);

```

```

            if (height > 0) {

```

```

                now.add(pre);

```

```

                now.add(id);

```

```

                now.add(height);

```

```

                ans.add(now);

```

```

            }

```

```

            pre = id;

```

```

            height = res.get(i).get(1);

```

```

        }

```

```

    }

```

```

    return ans;

```

```

}

```

```

public ArrayList<ArrayList<Integer>> buildingOutline(int[][][] buildings) {

```

```
// write your code here
ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<
Integer>>();

if (buildings == null || buildings.length == 0
    || buildings[0].length == 0) {
    return res;
}
ArrayList<Edge> edges = new ArrayList<Edge>();
for (int[] building : buildings) {
    Edge startEdge = new Edge(building[0], building[2], true);
    edges.add(startEdge);
    Edge endEdge = new Edge(building[1], building[2], false);
    edges.add(endEdge);
}
Collections.sort(edges, new EdgeComparator());

HashHeap heap = new HashHeap("max");

ArrayList<Integer> now = null;
for (Edge edge : edges) {
    if (edge.isStart) {
        if (heap.isEmpty() || edge.height > heap.peek()) {
            now = new ArrayList<Integer>(Arrays.asList(edge.pos,
                edge.height));
            res.add(now);
        }
        heap.add(edge.height);
    } else {
        heap.delete(edge.height);
        if (heap.isEmpty() || edge.height > heap.peek()) {
            if (heap.isEmpty()) {
                now = new ArrayList<Integer>(Arrays.asList(edge.pos,
0));
            } else {
                now = new ArrayList<Integer>(Arrays.asList(edge.pos,
                    heap.peek()));
            }
            res.add(now);
        }
    }
}
```

```
    }  
    }  
    return output(res);  
}  
  
}
```

# Sliding Window Median 360

## Question

Given an array of  $n$  integer, and a moving window(size  $k$ ), move the window at each iteration from the start of the array, find the median of the element inside the window at each moving. (If there are even numbers in the array, return the  $N/2$ -th number after sorting the element in the window. )

Example

For array  $[1,2,7,8,5]$ , moving window size  $k = 3$ . return  $[2,7,7]$

At first the window is at the start of the array like this

$[ | 1,2,7 | ,8,5 ]$  , return the median 2;

then the window move one step forward.

$[1, | 2,7,8 | ,5]$ , return the median 7;

then the window move one step forward again.

$[1,2, | 7,8,5 | ]$ , return the median 7;

Challenge

$O(n\log(n))$  time

## Solution

这道题和Data Stream Median类似，也是寻找动态数组的media，因此也可以用maxheap+minheap来解。随着窗口的移动，每次先加入一个新元素，再删去一个旧元素，再使两个heap中元素数量平衡即可。

1. 用一个最大堆来记录较小一半的元素，一个最小堆来记录较大一半的元素。

2. 先初始化最初的窗口里的k个元素。将元素加入最大堆，若为奇数次，则比较最大堆堆顶和最小堆堆顶元素大小，若最大堆堆顶元素大于和最小堆堆顶元素，则交换两个堆顶元素，若为偶数次，则将最大堆堆顶元素加入最小堆。
3. 然后开始移动窗口，先将新元素加入，若比原median小，则加入最大堆，反之则加入最小堆。然后删除窗口最前面的一个元素。
4. 然后调整最大堆和最小堆的大小。根据k值可以分两种情况讨论：
  - 1) 若k为偶数，则需要最大堆中元素和最小堆中元素数量相等
  - 2) 若k为奇数，则需要最大堆中元素比最小堆中元素多1个
5. 此时最大堆的堆顶元素即为此时窗口元素的median

代码如下：

```
public class Solution {
    /**
     * @param nums: A list of integers.
     * @return: The median of the element inside the window at each moving.
     */
    class Node{
        int val;
        int index;
        public Node(int val, int index){
            this.val = val;
            this.index = index;
        }
    }
    public ArrayList<Integer> medianSlidingWindow(int[] nums, int k) {
        // write your code here
        ArrayList<Integer> res = new ArrayList<Integer>();

        if(nums == null || nums.length < k || k < 1){
            return res;
        }

        Queue<Integer> max = new PriorityQueue<Integer>(1, new C
```

```
Comparator<Integer>(){
    public int compare(Integer a, Integer b){
        return b - a;
    }
});

Queue<Integer> min = new PriorityQueue<Integer>(1, new C
Comparator<Integer>(){
    public int compare(Integer a, Integer b){
        return a - b;
    }
});

//初始化第1个窗口
for(int i = 0; i < k; i++){
    max.offer(nums[i]);
    if(i % 2 == 0){
        if(min.size() != 0 && max.peek() > min.peek()){
            int maxTop = max.poll();
            int minTop = min.poll();
            min.offer(maxTop);
            max.offer(minTop);
        }
    }else{
        min.offer(max.poll());
    }
}
res.add(max.peek());

for(int i = k; i < nums.length; i++){
    //add
    if(nums[i] > max.peek()){
        min.offer(nums[i]);
    }else{
        max.offer(nums[i]);
    }
}

//delete，可以用HashHeap优化
if(max.contains(nums[i - k])){
    max.remove(nums[i - k]);
}
```

```
        }else{
            min.remove(nums[i - k]);
        }

        if(k % 2 == 0 && max.size() != min.size()){
            if(max.size() < min.size()){
                while(max.size() != min.size()){
                    max.offer(min.poll());
                }
            }else{
                while(max.size() != min.size()){
                    min.offer(max.poll());
                }
            }
        }




        if(k % 2 == 1 && max.size() != min.size() + 1){
            if(max.size() < min.size() + 1){
                while(max.size() != min.size() + 1){
                    max.offer(min.poll());
                }
            }else{
                while(max.size() != min.size() + 1){
                    min.offer(max.poll());
                }
            }
        }

        res.add(max.peek());
    }

    return res;
}
}
```



# Priority Queue

<div>⌵ Filter ⌵</div>	<div>⌵ Question No. ⌵</div>	<div>⚡ Pick one for me</div>
Medium	401. Kth Smallest Number in Sorted Matrix	✓ 19 %
Medium	104. Merge k Sorted Lists	  ✓ 27 %
Hard	81. Data Stream Median	 ✓ 26 %
Medium	4. Ugly Number II	✓ 22 %

# Ugly Number II 4

## Question

Ugly number is a number that only have factors 2, 3 and 5.

Design an algorithm to find the nth ugly number. The first 10 ugly numbers are 1, 2, 3, 4, 5, 6, 8, 9, 10, 12...

Notice

Note that 1 is typically treated as an ugly number.

Example

If  $n=9$ , return 10.

Challenge

$O(n \log n)$  or  $O(n)$  time.

## Solution

方法一：HashMap + PriorityQueue(MinHeap)：将2，3，5入队，每次取出堆顶元素（最小值），依次和3个factor相乘，将所得结果入队。用HashMap记录入队元素，入过队的元素不能再次入队。

方法二：以1为第一个元素，寻找比当前元素大的元素里面的最小者入队。每个factor用一个count来记录队列中第几个元素乘以factor以后恰好比current大，取3个factor结果里面的最小者作为下一个元素入队。

代码如下：

HashMap + Heap  $O(n \log n)$ ：

```
class Solution {  
    /**  
     * @param n an integer  
     * @return the nth prime number as description.  
     */  
}
```

```

    */
    public int nthUglyNumber(int n) {
        // Write your code here
        //HashMap + Heap O(nlogn)

        if(n <= 0){
            return 0;
        }

        PriorityQueue<Long> Q = new PriorityQueue<Long>();
        HashMap<Long, Boolean> inQ = new HashMap<Long, Boolean>(
);

        Long[] prime = new Long[3];
        prime[0] = Long.valueOf(2);
        prime[1] = Long.valueOf(3);
        prime[2] = Long.valueOf(5);
        for(int i = 0; i < 3; i++){
            Q.add(prime[i]);
            inQ.put(prime[i], true);
        }

        //每次取最小堆（优先队列）顶的元素，取n-1次，同时将取出的数和3个factor相乘的结果加入队列（如果本来已经在队列中则不用加）
        Long number = Long.valueOf(1);
        for(int i = 1; i < n; i++){
            number = Q.poll();
            for(int j = 0; j < 3; j++){
                if(!inQ.containsKey(number * prime[j])){
                    Q.add(number * prime[j]);
                    inQ.put(number * prime[j], true);
                }
            }
        }
        //Long转换为Int
        return number.intValue();
    }
}

```

O(n):

```
class Solution {
    /**
     * @param n an integer
     * @return the nth prime number as description.
     */
    public int nthUglyNumber(int n) {
        // Write your code here
        //O(n)
        ArrayList<Integer> ugly = new ArrayList<Integer>();
        ugly.add(1);
        int p1 = 0, p2 = 0, p3 = 0;
        int min1 = 0, min2 = 0, min3 = 0;
        int current = 1;

        //用current记录当前元素，寻找比当前元素大的数里面的最小值作为下一个元素
        while(ugly.size() < n){
            while(ugly.get(p1) * 2 < current){
                p1++;
            }
            min1 = ugly.get(p1) * 2;

            while(ugly.get(p2) * 3 < current){
                p2++;
            }
            min2 = ugly.get(p2) * 3;

            while(ugly.get(p3) * 5 < current){
                p3++;
            }
            min3 = ugly.get(p3) * 5;

            int next = Math.min(min1, min2);
            next = Math.min(next, min3);

            current = next + 1;
            ugly.add(next);
        }

        return ugly.get(n - 1);
    }
}
```

```
    }  
}
```

# Data Stream Median 81

## Question

Numbers keep coming, return the median of numbers at every time a new number added.

Clarification

What's the definition of Median?

- Median is the number that in the middle of a sorted array. If there are  $n$  numbers in a sorted array  $A$ , the median is  $A[(n - 1) / 2]$ . For example, if  $A = [1, 2, 3]$ , median is 2. If  $A = [1, 19]$ , median is 1.

Example

For numbers coming list:  $[1, 2, 3, 4, 5]$ , return  $[1, 1, 2, 2, 3]$ .

For numbers coming list:  $[4, 5, 1, 3, 2, 6, 0]$ , return  $[4, 4, 4, 3, 3, 3, 3]$ .

For numbers coming list:  $[2, 20, 100]$ , return  $[2, 2, 20]$ .

Challenge

Total run time in  $O(n \log n)$ .

## Solution

用一个最大堆（MaxHeap）来存到达的有所元素中较小的一半，用一个最小堆（MinHeap）来存到达的所有元素中较大的一半。每次来一个新的元素就先加入MaxHeap，偶数次加入MaxHeap（不变），奇数次将MaxHeap堆顶元素（最大值）移到MinHeap中。在偶数次时，若MaxHeap堆顶元素（较小一半的最大值）比MinHeap堆顶元素（较大一半的最小值）大，则交换两者位置。median为MaxHeap堆顶元素。

代码如下：

```
public class Solution {
    /**
     * @param nums: A list of integers.
     * @return: the median of numbers
     */
    private int numberOfElement = 0;
    PriorityQueue<Integer> maxHeap;
    PriorityQueue<Integer> minHeap;

    public int[] medianII(int[] nums) {
        // write your code here
        if(nums == null || nums.length <= 1){
            return nums;
        }
        //重构比较方法，使其从大到小排
        Comparator<Integer> revComp = new Comparator<Integer>(){
            public int compare(Integer left, Integer right){
                return right.compareTo(left);
            }
        };

        int length = nums.length;
        //维护一个最大堆和一个最小堆。
        maxHeap = new PriorityQueue<Integer>(length, revComp);
        minHeap = new PriorityQueue<Integer>(length);

        int[] result = new int[length];
        for(int i = 0; i < length; i++){
            addNumber(nums[i]);
            result[i] = getMedian();
        }

        return result;
    }

    private void addNumber(int value){
        maxHeap.add(value);
        if(numberOfElement % 2 == 0){
            if(minHeap.isEmpty()){
```

```
        numberOfElement++;
        return;
    }else{
        if(maxHeap.peek() > minHeap.peek()){
            Integer maxHeapRoot = maxHeap.poll();
            Integer minHeapRoot = minHeap.poll();
            maxHeap.add(minHeapRoot);
            minHeap.add(maxHeapRoot);
        }
    }
    }else{
        minHeap.add(maxHeap.poll());
    }
    numberOfElement++;
}

private int getMedian(){
    return maxHeap.peek();
}
}
```



# Trie

<div>Filter</div>	<div>Question No.</div>	<div>Pick one for me</div>
Medium	473. Add and Search Word	20 %
Medium	442. Implement Trie	<div><div></div><div></div></div> 27 %
Hard	132. Word Search II	<div><div></div><div></div></div> 20 %

## Word Search II 132

### Question

Given a matrix of lower alphabets and a dictionary. Find all words in the dictionary that can be found in the matrix. A word can start from any position in the matrix and go left/right/up/down to the adjacent position.

Example

Given matrix:

doaf

agai

dcan

and dictionary:

{"dog", "dad", "dgdg", "can", "again"}

return {"dog", "dad", "can", "again"}

dog:

```
doaf  
agai  
dcan
```

dad:

```
doaf  
agai  
dcan
```

can:

```
doaf  
agai  
dcan
```

again:

```
doaf  
agai  
dcan
```

### Challenge

Using trie to implement your algorithm.

## Solution

用Trie结构来解题。

Trie树，又称单词查找树、字典树，是一种树形结构，是一种哈希树的变种，是一种用于快速检索的多叉树结构。典型应用是用于统计和排序大量的字符串（但不仅限于字符串），所以经常被搜索引擎系统用于文本词频统计。它的优点是：最大限度地减少无谓的字符串比较，查询效率比哈希表高。Trie的核心思想是空间换时间。利用字符串的公共前缀来降低查询时间的开销以达到提高效率的目的。

三个基本特性：

- 1) 根节点不包含字符，除根节点外每一个节点都只包含一个字符。
- 2) 从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串。
- 3) 每个节点的所有子节点包含的字符都不相同。

首先用TreeNode和TrieTree两个class来构建Trie结构。将words中所有word都加入TrieTree中生成字典树，叶节点保存从root到叶节点路径组成的字符串。然后对board中字符逐个扫描，看以该字符开头的单词是否能在TrieTree中找到。search方法递归地寻找当前字符邻接的字符是否在TrieTree中当前字符对应TreeNode的子树中。直到叶节点时，若该字符串未被加入过，则将该字符串加入答案链表。

代码如下：

```
public class Solution {
    /**
     * @param board: A list of lists of character
     * @param words: A list of string
     * @return: A list of string
     */
    //树节点包含isString，s和subTree属性。isString和s在节点为叶节点时保存字符串信息，subTree记录该节点的所有子节点（即该字符之后所有可能字符）。
    class TreeNode{
        boolean isString;
        String s;
        HashMap<Character, TreeNode> subTree;
        public TreeNode(){
            isString = false;
            s = "";
            subTree = new HashMap<Character, TreeNode>();
        }
    }
    //TrieTree包含root属性和insert，find方法。root为根节点。insert方
```

法用于将一个新的字符串插入TrieTree，find方法用于判断一个字符串是否在TrieTree中。

```

class TrieTree{
    TreeNode root;
    public TrieTree(TreeNode root){
        this.root = root;
    }

    public void insert(String word){
        TreeNode now = root;
        for(int i = 0; i < word.length(); i++){
            if(!now.subTree.containsKey(word.charAt(i))){
                now.subTree.put(word.charAt(i), new TreeNode
());
            }
            now = now.subTree.get(word.charAt(i));
        }
        //叶节点纪录该字符串并将isString赋值为true
        now.isString = true;
        now.s = word;
    }

    public boolean find(String word){
        TreeNode now = root;
        for(int i = 0; i < word.length(); i++){
            if(!now.subTree.containsKey(word.charAt(i))){
                return false;
            }
        }
        return now.isString;
    }
}

//下右上左
private int[] dx = {1, 0, -1, 0};
private int[] dy = {0, 1, 0, -1};

private void search(int x, int y, char[][] board, TreeNode root, ArrayList<String> ans){
    //到达叶节点，判断该路径节点组成的字符串是否要加入答案链表。

```

```

        if(root.isString){
            if(!ans.contains(root.s)){
                ans.add(root.s);
            }
        }

        if(x < 0 || y < 0 || x >= board.length || y >= board[0].length || root == null){
            return;
        }

        //搜索当前字符四周字符，在搜索前将当前字符用'#'覆盖，以防重复搜索，在搜索完成后将其覆盖回原值。
        if(root.subTree.containsKey(board[x][y])){
            char curt = board[x][y];
            board[x][y] = '#';
            for(int i = 0; i < 4; i++){
                search(x + dx[i], y + dy[i], board, root.subTree.get(curt), ans);
            }
            board[x][y] = curt;
        }
    }

    public ArrayList<String> wordSearchII(char[][] board, ArrayList<String> words) {
        // write your code here
        ArrayList<String> ans = new ArrayList<String>();

        TrieTree trie = new TrieTree(new TreeNode());
        //构建TrieTree
        for(int i = 0; i < words.size(); i++){
            trie.insert(words.get(i));
        }
        //对board中字符逐个搜索
        for(int i = 0; i < board.length; i++){
            for(int j = 0; j < board[0].length; j++){
                search(i, j, board, trie.root, ans);
            }
        }
    }

```

```
        return ans;  
    }  
}
```

# Implement Trie 442

## Question

Implement a trie with insert, search, and startsWith methods.

Notice

You may assume that all inputs are consist of lowercase letters a-z.

Example

```
insert("lintcode")  
  
search("code") // return false  
  
startsWith("lint") // return true  
  
startsWith("lnterror") // return false  
  
insert("lnterror")  
  
search("lintcode") // return true  
  
startsWith("lnterror") // return true
```

## Solution

**TrieNode class**：中用一个HashMap来记录紧跟该node之后的所有node（每个字符都是一个node，即该字符之后所有可能字符）。若该node是叶节点，则将从root到该node路径上所有字符组成的字符串保存在该node的s中，并标记该路径是一个string。

**insert()**：将字符串插入Trie中。依次取出每一位字符，从root开始，若当前node之后的node不包含该字符，则新建一个该字符的node并加入当前node的suntree中，然后将当前node移往新建的node，循环直到字符串尾。



**search ()** : 搜索Trie中是否含有该字符串。依次取出每一位字符，从root开始，若当前node之后的node不包含该字符，则返回false，直到字符串尾。此时，若当前节点为叶节点，则返回true，否则返回false。

**startsWith ()** : 搜索Trie中是否含有以prefix为开头的字符串。方法和search () 一样，最后不用判断是否当前节点为叶节点。

代码如下：

```
/**
 * Your Trie object will be instantiated and called as such:
 * Trie trie = new Trie();
 * trie.insert("lintcode");
 * trie.search("lint"); will return false
 * trie.startsWith("lint"); will return true
 */
class TrieNode {
    String s;
    boolean isString;
    HashMap<Character, TrieNode> subTree;
    // Initialize your data structure here.
    public TrieNode() {
        s = "";
        isString = false;
        subTree = new HashMap<Character, TrieNode>();
    }
}

public class Trie {
    private TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    // Inserts a word into the trie.
    public void insert(String word) {
        if(word == null){
            return;
        }
    }
}
```

```
    TrieNode now = root;
    for(int i = 0; i < word.length(); i++){
        if(!now.subTree.containsKey(word.charAt(i))){
            now.subTree.put(word.charAt(i), new TrieNode());
        }
        now = now.subTree.get(word.charAt(i));
    }
    now.s = word;
    now.isString = true;
}

// Returns if the word is in the trie.
public boolean search(String word) {
    if(word == null){
        return true;
    }

    TrieNode now = root;
    for(int i = 0; i < word.length(); i++){
        if(!now.subTree.containsKey(word.charAt(i))){
            return false;
        }
        now = now.subTree.get(word.charAt(i));
    }

    if(now.isString){
        return true;
    }
    return false;
}

// Returns if there is any word in the trie
// that starts with the given prefix.
public boolean startsWith(String prefix) {
    if(prefix == null){
        return true;
    }

    TrieNode now = root;
```

```
        for(int i = 0; i < prefix.length(); i++){
            if(!now.subTree.containsKey(prefix.charAt(i))){
                return false;
            }
            now = now.subTree.get(prefix.charAt(i));
        }
        return true;
    }
}
```

# Add and Search Word 473

## Question

Design a data structure that supports the following two operations:

`addWord(word)` and `search(word)`

`search(word)` can search a literal word or a regular expression string containing only letters a-z or ..

A . means it can represent any one letter.

Notice

You may assume that all words are consist of lowercase letters a-z.

Example

```
addWord("bad")
```

```
addWord("dad")
```

```
addWord("mad")
```

```
search("pad") // return false
```

```
search("bad") // return true
```

```
search(".ad") // return true
```

```
search("b..") // return true
```

## Solution

1) 创建标准的TrieNode

2) `addWord()`: 和标准的Trie的`insert()`相同

3) `search()`: 本来只要依次比较`now.subTree`中是否含有当前字符，但是因为输入字符串中可能含有'.'(代表任意字符)，所以在遇到'.'时，需要查看`now.subTree`中包含的所有可能字符，即要查看从`now`这个点之后所有分岔路径是否能组成输入字符串。因此，迭代的方法不行，需要使用递归的方法。

递归检查当前节点`now`的`subtree`是否包含当前位置的字符，若包含则递归检查下一位置字符；若不包含，则看当前位置的字符是否为'.'，若不是则返回`false`，若是则依次递归检查当前节点`now`的`subtree`中包含的所有节点和下一位置字符，只要`subtree`中有一个返回`true`则返回`true`，否则返回`false`。

当检查字符串最后一个位置时，递归到底。此时若最后一位字符为'.'，则只要`now`的`subtree`中包含任意叶节点，即返回`true`，否则返回`false`。若最后一位字符不为'.'，若`now`的`subtree`包含该字符且该字符的节点为叶节点时，返回`true`，否则返回`false`。

代码如下：

```
class TrieNode{
    String s;
    boolean isString;
    HashMap<Character, TrieNode> subTree;
    public TrieNode(){
        s = "";
        isString = false;
        subTree = new HashMap<Character, TrieNode>();
    }
}

public class WordDictionary {
    TrieNode root = new TrieNode();
    // Adds a word into the data structure.
    public void addWord(String word) {
        // Write your code here
        if(word == null){
            return;
        }

        TrieNode now = root;
        for(int i = 0; i < word.length(); i++){
            if(!now.subTree.containsKey(word.charAt(i))){
```

```

        now.subTree.put(word.charAt(i), new TrieNode());
    }
    now = now.subTree.get(word.charAt(i));
}
now.s = word;
now.isString = true;
}

// Returns if the word is in the data structure. A word could
// contain the dot character '.' to represent any one letter
.
public boolean search(String word) {
    // Write your code here
    if(word == null){
        return true;
    }

    TrieNode now = root;
    return helper(now, word, 0);
}

private boolean helper(TrieNode now, String word, int pos){
    //递归到底时，若最后一位字符为'.', 则只要now的subtree中包含任意
    叶节点，即返回true，否则返回false
    if(pos == word.length() - 1){
        if(word.charAt(pos) == '.'){
            for(char c : now.subTree.keySet()){
                TrieNode next = now.subTree.get(c);
                if(next.isString){
                    return true;
                }
            }
            return false;
        }

        //若最后一位字符不为'.', 若now的subtree包含该字符且该字符的
        节点为叶节点时，返回true，否则返回false
        if(now.subTree.containsKey(word.charAt(pos)) && now.
subTree.get(word.charAt(pos)).isString){

```

```

        return true;
    }

    return false;
}

//递归检查当前节点now的subtree是否包含当前位置的字符，若包含则递归检查下一位置字符
if(now.subTree.containsKey(word.charAt(pos))){
    return helper(now.subTree.get(word.charAt(pos)), word, pos + 1);
}else{
    //若不包含，则看当前位置的字符是否为'.',若是则依次递归检查当前节点now的subtree中包含的所有节点和下一位置字符，只要subtree中有一个返回true则返回true，否则返回false
    if(word.charAt(pos) == '.'){
        ArrayList<Boolean> hasString = new ArrayList<Boolean>();

        for(char c : now.subTree.keySet()){
            TrieNode next = now.subTree.get(c);
            hasString.add(helper(next, word, pos + 1));
        }
        for(boolean b : hasString){
            if(b){
                return true;
            }
        }
        return false;
    }

    return false;
}
}
}

```

```

// Your WordDictionary object will be instantiated and called as such:
// WordDictionary wordDictionary = new WordDictionary();
// wordDictionary.addWord("word");
// wordDictionary.search("pattern");

```





# Binary Tree

## Subtree 245

### Question

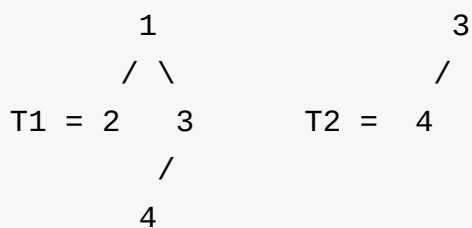
You have two very large binary trees: T1, with millions of nodes, and T2, with hundreds of nodes. Create an algorithm to decide if T2 is a subtree of T1.

Notice

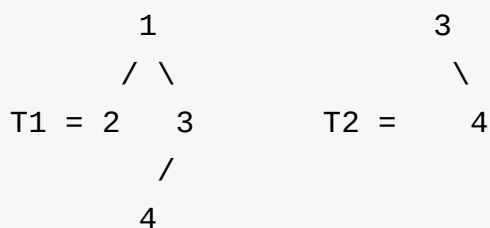
A tree T2 is a subtree of T1 if there exists a node n in T1 such that the subtree of n is identical to T2. That is, if you cut off the tree at node n, the two trees would be identical.

Example

T2 is a subtree of T1 in the following case:



T2 isn't a subtree of T1 in the following case:



### Solution

非递归版本：非递归版本用BFS遍历T1,找所有与T2值相等的点，作为备选节点。然后比较以这些节点为根的子树是否和T2等同。

递归版本：先看T1和T2是否等同，若不同，则递归看T1的左右子树是否和T2等同。

代码如下：

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param T1, T2: The roots of binary tree.
     * @return: True if T2 is a subtree of T1, or false.
     */
    public boolean isSubtree(TreeNode T1, TreeNode T2) {
        // write your code here
        if(T2 == null){
            return true;
        }

        if(T1 == null){
            return false;
        }

        //非递归 Version
        // Queue<TreeNode> queue = new LinkedList<TreeNode>();
        // queue.offer(T1);

        // TreeNode root = new TreeNode(T2.val + 1);
        // ArrayList<TreeNode> candidates = new ArrayList<TreeNo
de>();

        // while(!queue.isEmpty()){
```

```
//      TreeNode curt = queue.poll();
//      if(curt.val == T2.val){
//          root = curt;
//          candidates.add(curt);
//      }
//      if(curt.left != null){
//          queue.offer(curt.left);
//      }
//      if(curt.right != null){
//          queue.offer(curt.right);
//      }
//  }

// if(root.val != T2.val){
//     return false;
// }

// for(TreeNode node : candidates){
//     if(isIdentical(node, T2)){
//         return true;
//     }
// }

// return false;

//递归 Version
if(isIdentical(T1, T2)){
    return true;
}

if(isSubtree(T1.left, T2) || isSubtree(T1.right, T2)){
    return true;
}

return false;
}

private boolean isIdentical(TreeNode a, TreeNode b){
    if(a == null || b == null){
        return a == b;
    }
}
```

```
    }

    if(a.val != b.val){
        return false;
    }

    return isIdentical(a.left, b.left) && isIdentical(a.right, b.right);
}
}
```

# Flatten Binary Tree to Linked List 453

## Question

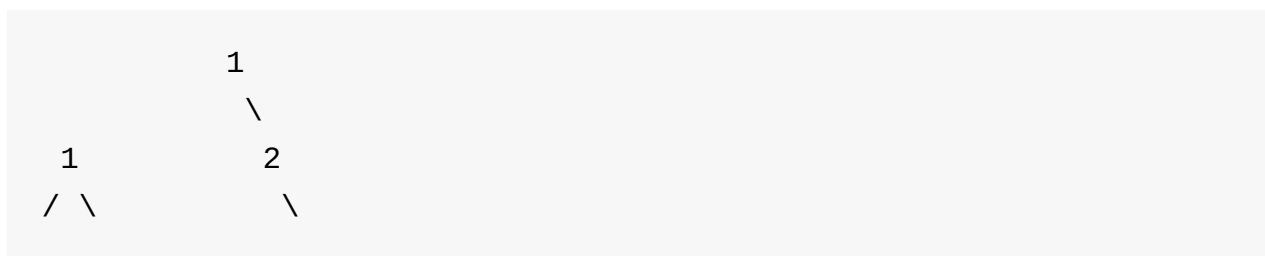
Flatten a binary tree to a fake "linked list" in pre-order traversal.

Here we use the right pointer in `TreeNode` as the next pointer in `ListNode`.

Notice

Don't forget to mark the left child of each node to null. Or you will get Time Limit Exceeded or Memory Limit Exceeded.

Example



2 5 => 3 / \ \ \ 3 4 6 4 \ 5 \ 6 Challenge

Do it in-place without any extra memory.

## Solution

递归方法：Divide and Conquer.

非递归方法：用stack实现。将root加入stack后，出栈时，现将right加入，再将left加入。然后将出栈元素的left设为null，如果此时stack为空（即出栈元素为最后一个元素），则出栈元素的right也设为null；如果不为空，则将栈顶元素（即出栈元素的原left）设为出栈元素的right。

代码如下：

Recursion Version

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param root: a TreeNode, the root of the binary tree
     * @return: nothing
     */
    public void flatten(TreeNode root) {
        // write your code here

        helper(root);

        private TreeNode helper(TreeNode root){
            if(root == null || (root.left == null && root.right == null)){
                return root;
            }

            if(root.left == null){
                return helper(root.right);
            }

            if(root.right == null){
                root.right = root.left;
                root.left = null;
                return helper(root.right);
            }

            TreeNode leftLast = helper(root.left);
```

```

        TreeNode rightLast = helper(root.right);

        leftLast.right = root.right;
        root.right = root.left;
        root.left = null;
        return rightLast;
    }
}

```

### Non-Recursion Version

```

/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param root: a TreeNode, the root of the binary tree
     * @return: nothing
     */
    public void flatten(TreeNode root) {
        // write your code here
        //Non-Recursion Version
        if(root == null || (root.left == null && root.right == null)){
            return;
        }

        Stack<TreeNode> stack = new Stack<TreeNode>();
        stack.push(root);
        while(!stack.isEmpty()){
            TreeNode curt = stack.pop();

```



```
        if(curt.right != null){
            stack.push(curt.right);
        }

        if(curt.left != null){
            stack.push(curt.left);
        }

        curt.left = null;
        if(stack.isEmpty()){
            curt.right = null;
        }else{
            curt.right = stack.peek();
        }
    }
}
```

# Binary Tree Paths 480

## Question

Given a binary tree, return all root-to-leaf paths.

Example

Given the following binary tree:

1 / \ 2 3 \ 5

All root-to-leaf paths are:

[ "1->2->5",  
"1->3" ]

## Solution

Divide and Conquer.

九章给的答案是recursion.

代码如下：

D&C:

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
```

```
public class Solution {
    /**
     * @param root the root of the binary tree
     * @return all root-to-leaf paths
     */
    public List<String> binaryTreePaths(TreeNode root) {
        // Write your code here
        List<String> result = new ArrayList<String>();

        if(root == null){
            return result;
        }

        if(root.left == null && root.right == null){
            result.add(String.valueOf(root.val));
        }

        List<String> left = binaryTreePaths(root.left);
        List<String> right = binaryTreePaths(root.right);

        if(left.size() != 0){
            for(String s : left){
                StringBuilder sb = new StringBuilder();
                sb.append(root.val);
                sb.append("->");
                sb.append(s);
                result.add(sb.toString());
            }
        }

        if(right.size() != 0){
            for(String s : right){
                StringBuilder sb = new StringBuilder();
                sb.append(root.val);
                sb.append("->");
                sb.append(s);
                result.add(sb.toString());
            }
        }
    }
}
```

```
        return result;  
    }  
}
```

Recursion:

```
public class Solution {
    /**
     * @param root the root of the binary tree
     * @return all root-to-leaf paths
     */
    public List<String> binaryTreePaths(TreeNode root) {
        List<String> result = new ArrayList<String>();
        if (root == null) {
            return result;
        }
        helper(root, String.valueOf(root.val), result);
        return result;
    }

    private void helper(TreeNode root, String path, List<String>
result) {
        if (root == null) {
            return;
        }

        if (root.left == null && root.right == null) {
            result.add(path);
            return;
        }

        if (root.left != null) {
            helper(root.left, path + "->" + String.valueOf(root.
left.val), result);
        }

        if (root.right != null) {
            helper(root.right, path + "->" + String.valueOf(root
.right.val), result);
        }
    }
}
```



# Binary Tree Maximum Path Sum II 475

## Question

Given a binary tree, find the maximum path sum from root.

The path may end at any node in the tree and contain at least one node in it.

Example

Given the below binary tree:

1 / \ 2 3

return 4. (1->3)

## Solution

水题，直接通d&c即可。唯一要注意的是题目要求至少有一个点，所以左右值先和0比较后再和root值相加。

代码如下：

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param root the root of binary tree.
     * @return an integer
     */
    public int maxPathSum2(TreeNode root) {
        // Write your code here
        if(root == null){
            return 0;
        }

        int left = maxPathSum2(root.left);
        int right = maxPathSum2(root.right);

        return root.val + Math.max(0, Math.max(left, right));
    }
}
```



# Union Find

# Graph Valid Tree 178

## Question

Given  $n$  nodes labeled from 0 to  $n - 1$  and a list of undirected edges (each edge is a pair of nodes), write a function to check whether these edges make up a valid tree.

Notice

You can assume that no duplicate edges will appear in edges. Since all edges are undirected,  $[0, 1]$  is the same as  $[1, 0]$  and thus will not appear together in edges.

Example

Given  $n = 5$  and edges =  $[[0, 1], [0, 2], [0, 3], [1, 4]]$ , return true.

Given  $n = 5$  and edges =  $[[0, 1], [1, 2], [2, 3], [1, 3], [1, 4]]$ , return false.

## Solution

首先tree必须保证 $n$ 个点只有 $n-1$ 条边。然后必须保证所有点在一个联通分量中，并且无环。有两种方法：

1. 用UF，将边上的两个点归并到同一个联通分量中，若已经在同一个分量中的两个点之间还有边，则有环
2. 从一个点开始，用BFS，若能遍历所有点则所有点在一个联通分量中

代码如下：

Union-Find version:

```
public class Solution {  
    /**  
     * @param n an integer  
     * @param edges a list of undirected edges  
     * @return true if it's a valid tree, or false  
     */  
}
```

```
*/
class UnionFind{
    HashMap<Integer, Integer> father = new HashMap<Integer,
Integer>();
    public UnionFind(int n){
        for(int i = 0; i < n; i++){
            father.put(i,i);
        }
    }

    public int compressed_find(int x){
        int parent = father.get(x);
        while(parent != father.get(parent)){
            parent = father.get(parent);
        }

        int temp = 0;
        int fa = x;
        while(fa != father.get(fa)){
            temp = father.get(fa);
            father.put(fa, parent);
            fa = temp;
        }

        return parent;
    }

    public void union(int a, int b){
        int aFather = compressed_find(a);
        int bFather = compressed_find(b);
        if(aFather != bFather){
            father.put(aFather, bFather);
        }
    }
}

public boolean validTree(int n, int[][] edges) {
    // Write your code here

    //UF version
    if(n != edges.length + 1){
```

```

        return false;
    }

    if(n == 1){
        return true;
    }

    UnionFind uf = new UnionFind(n);

    for(int i = 0; i < edges.length; i++){
        if(uf.compressed_find(edges[i][0]) == uf.compressed_
find(edges[i][1])){
            return false;
        }
        uf.union(edges[i][0], edges[i][1]);
    }

    return true;
}
}

```

BFS version:

```

public class Solution {
    /**
     * @param n an integer
     * @param edges a list of undirected edges
     * @return true if it's a valid tree, or false
     */
    public boolean validTree(int n, int[][] edges) {
        // Write your code here
        //bfs version
        if(n != edges.length + 1){
            return false;
        }

        if(n == 1){
            return true;
        }
    }
}

```

```
Queue<Integer> queue = new LinkedList<Integer>();
HashSet<Integer> set = new HashSet<Integer>();
queue.offer(0);
set.add(0);

while(!queue.isEmpty()){
    int curt = queue.poll();
    for(int i = 0; i < edges.length; i++){
        if(edges[i][0] == curt && !set.contains(edges[i]
[1])){
            queue.offer(edges[i][1]);
            set.add(edges[i][1]);
        }
        if(edges[i][1] == curt && !set.contains(edges[i]
[0])){
            queue.offer(edges[i][0]);
            set.add(edges[i][0]);
        }
    }
}

return set.size() == n;
}
```

# Surrounded Regions 477

## Question

Given a 2D board containing 'X' and 'O', capture all regions surrounded by 'X'.

A region is captured by flipping all 'O's into 'X's in that surrounded region.

Example

X X X X

X O O X

X X O X

X O X X

After capture all regions surrounded by 'X', the board should be:

X X X X

X X X X

X X X X

X O X X

## Solution

board中的'O'只能属于两类：1）边界的'O'（没有被'X'完全包围）；2）内部的'O'（被'X'完全包围）。因此，只要搜索board上下左右四条边界，找出所有边界'O'的union中的所有点，将其标为'F'，则board中剩下的'O'必然属于surround region。

代码如下：

```
public class Solution {  
    /**
```

```

    * @param board a 2D board containing 'X' and 'O'
    * @return void
    */
    //搜索board上下左右四个边界，找到其边界上'O'的union，将其标为'F'，则
    board中剩下的'O'必然属于surround region
    //上下左右搜索
    static final int[] dx = {-1, 1, 0, 0};
    static final int[] dy = {0, 0, -1, 1};
    //用F表明和边界'O'相连的'O'，用'V'表明访问过的点
    static final char Free = 'F';
    static final char Visited = 'V';

    class Node{
        int x;
        int y;
        public Node(int x, int y){
            this.x = x;
            this.y = y;
        }
    }

    public void surroundedRegions(char[][] board) {
        // Write your code here
        if(board == null || board.length == 0 || board[0].length
        == 0){
            return;
        }

        int row = board.length;
        int col = board[0].length;
        //找出左右边界上'O'的union，并将其都替换为'F'
        for(int i = 0; i < row; i++){
            bfs(board, i, 0);
            bfs(board, i, col - 1);
        }
        //找出上下边界上'O'的union，并将其都替换为'F'
        for(int j = 1; j < col - 1; j++){
            bfs(board, 0, j);
            bfs(board, row - 1, j);
        }
    }

```

//遍历整个board，所有标记为'F'的点都属于边界上的'O'的union，将其都变回原来的值；所有标记仍为'O'的点都属于surrounded regions，将其都变为'X'

```

    for(int i = 0; i < row; i++){
        for(int j = 0; j < col; j++){
            if(board[i][j] == 'O'){
                board[i][j] = 'X';
            }
            if(board[i][j] == 'F'){
                board[i][j] = 'O';
            }
        }
    }
}

```

```

private void bfs(char[][] board, int x, int y){
    if(board[x][y] == 'X'){
        return;
    }

```

//用bfs算法搜索所有边界上'O'的union的所有点并入列，在出列时将其变为'F'，以表示该点属于边界上'O'的union

```

    Queue<Node> queue = new LinkedList<Node>();
    queue.offer(new Node(x, y));

```

```

    while(!queue.isEmpty()){
        Node curt = queue.poll();
        board[curt.x][curt.y] = Free;

        for(Node node : expand(board, curt.x, curt.y)){
            queue.offer(node);
        }
    }
}

```

//搜索当前点四周的点，返回合法的点的list

```

private ArrayList<Node> expand(char[][] board, int x, int y)
{
    ArrayList<Node> expansion = new ArrayList<Node>();

    for(int i = 0; i < dx.length; i++){
        int curtX = x + dx[i];

```



```
        int curtY = y + dy[i];
        //当其周围的点不越界，同时为'0'时，入列
        if(curtX >= 0 && curtY >= 0 && curtX < board.length
&& curtY < board[0].length && (board[curtX][curtY] == '0')){
            //将入列的点标记为'V'，这样可以保证之后不会重复入列
            board[curtX][curtY] = Visited;
            expansion.add(new Node(curtX, curtY));
        }
    }

    return expansion;
}
}
```

# Number of Islands 433

## Question

Given a boolean 2D matrix, find the number of islands.

Notice

0 is represented as the sea, 1 is represented as the island. If two 1 is adjacent, we consider them in the same island. We only consider up/down/left/right adjacent.

Example

Given graph:

```
[  
[1, 1, 0, 0, 0],  
[0, 1, 0, 0, 1],  
[0, 0, 0, 1, 1],  
[0, 0, 0, 0, 0],  
[0, 0, 0, 0, 1]  
]  
return 3.
```

## Solution

这道题和Graph Valid Tree 178一样，既能用UF也能用dfs/bfs来寻找联通分量。

代码如下：

UF:

```
public class Solution {
```

```
/**
 * @param grid a boolean 2D matrix
 * @return an integer
 */
//uf模版
class UnionFind{
    HashMap<Integer, Integer> father = new HashMap<Integer,
Integer>();
    public UnionFind(int[] array){
        for(int i = 0; i < array.length; i++){
            father.put(i, i);
        }
    }

    public int compressed_find(int x){
        int parent = father.get(x);
        while(parent != father.get(parent)){
            parent = father.get(parent);
        }

        int temp = 0;
        int fa = x;
        while(fa != father.get(fa)){
            temp = father.get(fa);
            father.put(fa, parent);
            fa = temp;
        }

        return parent;
    }

    public void union(int a, int b){
        int aFather = compressed_find(a);
        int bFather = compressed_find(b);
        // if(aFather != bFather){
        //     father.put(aFather, bFather);
        // }
        father.put(aFather, bFather);
    }
}
```

```

int[] dx = {-1, 1, 0, 0};
int[] dy = {0, 0, -1, 1};
public int numIslands(boolean[][] grid) {
    // Write your code here
    if(grid == null || grid.length == 0 || grid[0].length ==
0){
        return 0;
    }

    int n = grid.length;
    int m = grid[0].length;
    int[] grid1D = new int[n * m];
    int count = 0;

    for(int i = 0; i < grid.length; i++){
        for(int j = 0; j < grid[0].length; j++){
            int index = i * m + j;
            if(grid[i][j] == true){
                grid1D[index] = 1;
                count++;
            }else{
                grid1D[index] = 0;
            }
        }
    }

    UnionFind uf = new UnionFind(grid1D);

    for(int i = 0; i < grid1D.length; i++){
        if(grid1D[i] == 0){
            continue;
        }

        for(int j = 0; j < 4; j++){
            int indexX = i / m + dx[j];
            int indexY = i % m + dy[j];
            int index = indexX * m + indexY;
            if(indexX >= 0 && indexX < n && indexY >= 0 && i
ndexY < m && grid1D[i] == grid1D[index]){

```

```
        if(uf.compressed_find(i) != uf.compressed_find(index)){
            uf.union(i, index);
            count--;
        }
    }
    return count;
}
```

DFS:

```
public class Solution {
    /**
     * @param grid a boolean 2D matrix
     * @return an integer
     */
    private int m, n;
    public void dfs(boolean[][] grid, int i, int j) {
        if (i < 0 || i >= m || j < 0 || j >= n) return;

        if (grid[i][j]) {
            grid[i][j] = false;
            dfs(grid, i - 1, j);
            dfs(grid, i + 1, j);
            dfs(grid, i, j - 1);
            dfs(grid, i, j + 1);
        }
    }

    public int numIslands(boolean[][] grid) {
        // Write your code here
        m = grid.length;
        if (m == 0) return 0;
        n = grid[0].length;
        if (n == 0) return 0;

        int ans = 0;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (!grid[i][j]) continue;
                ans++;
                dfs(grid, i, j);
            }
        }
        return ans;
    }
}
```



# Find the Connected Component in the Undirected Graph 431

## Question

Find the number connected component in the undirected graph. Each node in the graph contains a label and a list of its neighbors. (a connected component (or just component) of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph.)

Example

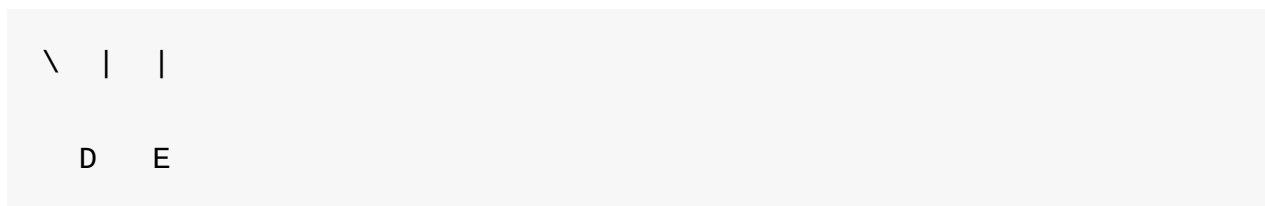
Given graph:

A-----B C

\ | |

\ | |

\ | |



Return {A,B,D}, {C,E}. Since there are two connected component which is {A,B,D}, {C,E}

## Solution

DFS:将任一点加入备选答案中，同时标记为已访问，对其任意未访问过的neighbor递归调用DFS，直到没有点能被加入备选答案为止，此时备选答案为一个CC。遍历输入的所有node，直到所有点都被访问过为止。



\* DFS容易造成递归过深的问题，例如所有node连成一条直线。

BFS:将任一点加入queue中，在该点出队时，将其加入备选答案中，同时将其所有未入队过的neighbor加入queue中，依此出队入队直到queue为空为止，此时备选答案为一个CC。遍历输入的所有node，直到所有点都入过队为止。

代码如下：

DFS:

```

**
 * Definition for Undirected graph.
 * class UndirectedGraphNode {
 *     int label;
 *     ArrayList<UndirectedGraphNode> neighbors;
 *     UndirectedGraphNode(int x) { label = x; neighbors = new A
rrayList<UndirectedGraphNode>(); }
 * }
 */
public class Solution {
    /**
     * @param nodes a array of Undirected graph node
     * @return a connected set of a Undirected graph
     */
    public List<List<Integer>> connectedSet(ArrayList<Undirected
GraphNode> nodes) {
        if (nodes == null || nodes.size() == 0) return null;

        List<List<Integer>> result = new ArrayList<List<Integer>
>();
        Set<UndirectedGraphNode> visited = new HashSet<Undirecte
dGraphNode>();
        for (UndirectedGraphNode node : nodes) {
            if (visited.contains(node)) continue;
            List<Integer> temp = new ArrayList<Integer>();
            dfs(node, visited, temp);
            Collections.sort(temp);
            result.add(temp);
        }
    }
}

```

```

        return result;
    }

    private void dfs(UndirectedGraphNode node,
                    Set<UndirectedGraphNode> visited,
                    List<Integer> result) {

        // add node into result
        result.add(node.label);
        visited.add(node);
        // node is not connected, exclude by for iteration
        // if (node.neighbors.size() == 0 ) return;
        for (UndirectedGraphNode neighbor : node.neighbors) {
            if (visited.contains(neighbor)) continue;
            dfs(neighbor, visited, result);
        }
    }
}

```

BFS:

```

/**
 * Definition for Undirected graph.
 * class DirectedGraphNode {
 *     int label;
 *     ArrayList<DirectedGraphNode> neighbors;
 *     UndirectedGraphNode(int x) { label = x; neighbors = new A
rrayList<DirectedGraphNode>(); }
 * };
 */

public List<List<Integer>> connectedSet(ArrayList<UndirectedGrap
hNode> nodes) {
    // Write your code here

    int m = nodes.size();
    Map<UndirectedGraphNode, Boolean> visited = new HashMap<
>();

```

```
        for (UndirectedGraphNode node : nodes){
            visited.put(node, false);
        }

        List<List<Integer>> result = new ArrayList<>();

        for (UndirectedGraphNode node : nodes){
            if (visited.get(node) == false){
                bfs(node, visited, result);
            }
        }

        return result;
    }

    public void bfs(UndirectedGraphNode node, Map<UndirectedGraph
hNode, Boolean> visited, List<List<Integer>> result){
        List<Integer> row = new ArrayList<Integer>();
        Queue<UndirectedGraphNode> queue = new LinkedList<>();
        visited.put(node, true);
        queue.offer(node);
        while (!queue.isEmpty()){
            UndirectedGraphNode u = queue.poll();
            row.add(u.label);
            for (UndirectedGraphNode v : u.neighbors){
                if (visited.get(v) == false){
                    visited.put(v, true);
                    queue.offer(v);
                }
            }
        }
        Collections.sort(row);
        result.add(row);
    }
}
```

# Find the Weak Connected Component in the Directed Graph 432

## Question

Find the number Weak Connected Component in the directed graph. Each node in the graph contains a label and a list of its neighbors. (a connected set of a directed graph is a subgraph in which any two vertices are connected by direct edge path.)

Notice

Sort the element in the set in increasing order

Example

Given graph:

```

A----->B  C
 \       |  |
  \       |  |
   \       |  |
    \      v  v
      ->D  E <- F
  
```

Return {A,B,D}, {C,E,F}. Since there are two connected component which are {A,B,D} and {C,E,F}

## Solution

用UF模板来解Connected Component问题。用路径压缩来优化搜索。

1. 初始化所有的点，parent为其自身
2. 遍历图中每个点及其neighbor，用uf找到图中所有cc，路径压缩使每个cc中的所有点都有唯一parent

3. 再次遍历图中所有点，用一个hashset记录图中找过的cc的代表元素。若某点所在cc还未被找过，则寻找所有该cc中的点加入list，同时将该cc的代表元素加入hashset
4. 重复3直到遍历所有点

代码如下：

```
/**
 * Definition for Directed graph.
 * class DirectedGraphNode {
 *     int label;
 *     ArrayList<DirectedGraphNode> neighbors;
 *     DirectedGraphNode(int x) { label = x; neighbors = new Arr
ArrayList<DirectedGraphNode>(); }
 * };
 */
public class Solution {
    /**
     * @param nodes a array of Directed graph node
     * @return a connected set of a directed graph
     */
    class UnionFind{
        HashMap<Integer, Integer> father = new HashMap<Integer,
Integer>();
        public UnionFind(ArrayList<Integer> array){
            for(int i : array){
                father.put(i, i);
            }
        }

        public int compressed_find(int x){
            int parent = father.get(x);
            while(parent != father.get(parent)){
                parent = father.get(parent);
            }

            int temp = 0;
            int fa = x;
            while(fa != father.get(fa)){
```

```
        temp = father.get(fa);
        father.put(fa, parent);
        fa = temp;
    }

    return parent;
}

public void union(int a, int b){
    int aFather = compressed_find(a);
    int bFather = compressed_find(b);
    if(aFather != bFather){
        father.put(aFather, bFather);
    }
}

}

public List<List<Integer>> connectedSet2(ArrayList<DirectedG
raphNode> nodes) {
    // Write your code here
    List<List<Integer>> result = new ArrayList<List<Integer>
>();
    ArrayList<Integer> array = new ArrayList<Integer>();
    for(DirectedGraphNode dgn : nodes){
        if(!array.contains(dgn.label)){
            array.add(dgn.label);
        }
        for(DirectedGraphNode neighbor : dgn.neighbors){
            if(!array.contains(neighbor.label)){
                array.add(neighbor.label);
            }
        }
    }

    UnionFind uf = new UnionFind(array);

    for(DirectedGraphNode dgn : nodes){
        for(DirectedGraphNode neighbor : dgn.neighbors){
            uf.union(dgn.label, neighbor.label);
        }
    }
}
```

```
        HashSet<Integer> set = new HashSet<Integer>();
        for(int i = 0; i < nodes.size(); i++){
            if(set.contains(uf.compressed_find(nodes.get(i).label))){
                continue;
            }

            set.add(uf.compressed_find(nodes.get(i).label));

            List<Integer> list = new ArrayList<Integer>();
            list.add(nodes.get(i).label);
            for(int j = i + 1; j < nodes.size(); j++){
                if(uf.compressed_find(nodes.get(j).label) == uf.compressed_find(nodes.get(i).label)){
                    list.add(nodes.get(j).label);
                }
            }
            result.add(list);
        }

        return result;
    }
}
```

# Number of Islands II 434

## Question

Given a  $n, m$  which means the row and column of the 2D matrix and an array of pair  $A$  (size  $k$ ). Originally, the 2D matrix is all 0 which means there is only sea in the matrix. The list pair has  $k$  operator and each operator has two integer  $A[i].x$ ,  $A[i].y$  means that you can change the grid matrix  $A[i].x[A[i].y]$  from sea to island. Return how many island are there in the matrix after each operator.

Notice

0 is represented as the sea, 1 is represented as the island. If two 1 is adjacent, we consider them in the same island. We only consider up/down/left/right adjacent.

Example

Given  $n = 3$ ,  $m = 3$ , array of pair  $A = [(0,0),(0,1),(2,2),(2,1)]$ .

return  $[1,1,2,2]$ .

## Solution

用UF模版来寻找图中联通分量的个数。用路径压缩来优化搜索。

1. 首先将二维数组转化为1维数组，初始化uf，并用一个数组来记录某一个位置是否被visit
2. 用一个count来记录每一次操作后联通分量的个数。每改变一个位置，就将count加1，并将该位置记录为visit。然后看该位置上下左右四个位置，若其相邻位置不越界且曾经被visit过，并且不属于一个联通分量，则将该位置和其相邻位置union，并将count减1。所有相邻位置看完后剩下的count即为本次操作后的联通分量数。

代码如下：

```
/**
```



```
* Definition for a point.
* class Point {
*     int x;
*     int y;
*     Point() { x = 0; y = 0; }
*     Point(int a, int b) { x = a; y = b; }
* }
*/

public class Solution {
    /**
     * @param n an integer
     * @param m an integer
     * @param operators an array of point
     * @return an integer array
     */
    //uf template
    class UnionFind{
        HashMap<Integer, Integer> father = new HashMap<Integer,
Integer>();
        public UnionFind(int[] array){
            for(int i : array){
                father.put(i, i);
            }
        }

        public int compressed_find(int x){
            int parent = father.get(x);
            while(parent != father.get(parent)){
                parent = father.get(parent);
            }

            int temp = 0;
            int fa = x;
            while(fa != father.get(fa)){
                temp = father.get(fa);
                father.put(fa, parent);
                fa = temp;
            }

            return parent;
        }
    }
}
```

```

    }

    public void union(int a, int b){
        int aFather = compressed_find(a);
        int bFather = compressed_find(b);
        if(aFather != bFather){
            father.put(aFather, bFather);
        }
    }
}

int[] dx = {-1, 1, 0, 0};
int[] dy = {0, 0, -1, 1};
public List<Integer> numIslands2(int n, int m, Point[] operators) {
    // Write your code here
    List<Integer> res = new ArrayList<Integer>();
    if(n < 1 || m < 1 || operators == null || operators.length == 0){
        return res;
    }

    int[] pos = new int[n * m];
    boolean[] isIsland = new boolean[n * m];

    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            int index = i * m + j;
            pos[index] = index;
        }
    }

    UnionFind uf = new UnionFind(pos);

    int count = 0;
    for(Point point : operators){
        int index = point.x * m + point.y;
        isIsland[index] = true;
        count++;
        for(int i = 0; i < 4; i++){

```

```
        int nextX = point.x + dx[i];
        int nextY = point.y + dy[i];
        int nextIndex = nextX * m + nextY;
        if(nextX >= 0 && nextX < n && nextY >= 0 && nextY < m && isIsland[nextIndex]){
            if(uf.compressed_find(index) != uf.compressed_find(nextIndex)){
                uf.union(index, nextIndex);
                count--;
            }
        }
        res.add(count);
    }

    return res;
}
```

# Binary Search Tree

# Validate Binary Search Tree 95

## Question

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

The left subtree of a node contains only nodes with keys less than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key. Both the left and right subtrees must also be binary search trees. A single node tree is a BST

Example

An example:

2 / \ 1 4 / \ 3 5

The above binary tree is serialized as {2,1,4,##,3,5} (in level order).

## Solution

用一个resulttype结构记录以某一节点为根的子树是否是bst以及其最大值和最小值。分：分别看左右子树是否为bst以及寻找其最大值和最小值。治：看左右子树是否都为bst并且左子树的最大值是否小于root的值，以及root的值是否小于右子树的最小值。

边界：当root==null时，1) 树的根节点为null，返回true，2) 叶子节点的左右子树根节点为null，返回true，此时max值赋值为无穷小，min值赋值为无穷大，这是因为最后一行比较root value和max，min时不会取到max和min的值

代码如下：

D & C:

```
/**
 * Definition of TreeNode:
```

```
* public class TreeNode {
*     public int val;
*     public TreeNode left, right;
*     public TreeNode(int val) {
*         this.val = val;
*         this.left = this.right = null;
*     }
* }
*/
class ResultType{
    public boolean isBST;
    public int maxValue;
    public int minValue;

    public ResultType(boolean isBST, int maxValue, int minValue)
    {
        this.isBST = isBST;
        this.maxValue = maxValue;
        this.minValue = minValue;
    }
}

public class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: True if the binary tree is BST, or false
     */
    public boolean isValidBST(TreeNode root) {
        // write your code here
        ResultType r = validateHelper(root);
        return r.isBST;
    }

    public ResultType validateHelper(TreeNode root){
        if(root == null){
            return new ResultType(true, Integer.MIN_VALUE, Integer.MAX_VALUE);
        }

        ResultType left = validateHelper(root.left);
```

```
        ResultType right = validateHelper(root.right);

        if(!left.isBST || !right.isBST){
            return new ResultType(false, 0, 0);
        }

        if((root.left != null && root.val <= left.maxValue) || (
root.right !=null && root.val >= right.minValue)){
            return new ResultType(false, 0, 0);
        }

        return new ResultType(true, Math.max(root.val, right.max
Value), Math.min(root.val, left.minValue));
    }
}
```

# Insert Node in a Binary Search Tree 85

## Question

Given a binary search tree and a new tree node, insert the node into the tree. You should keep the tree still be a valid binary search tree.

Notice

You can assume there is no duplicate values in this tree + node.

Example

Given binary search tree as follow, after Insert node 6, the tree should be:

2 2 /\ /\ 1 4 --> 1 4 /\ /\ 3 3 6

Challenge

Can you do it without recursion?

## Solution

Recursion: 若插入node值大于root值，则插入root右子树，若小于则插入root左子树

Non-Recursion: 用stack实现。若插入的node值大于root值，则将root加入stack，将root变为其右孩子，反之则变为其左孩子，直到root为null为止，此时stack的栈顶元素即为node要插入的元素，node值大于该元素则插在右边，否则插在左边。

代码如下：

Recursion:



```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param root: The root of the binary search tree.
     * @param node: insert this node into the binary search tree
     * @return: The root of the new binary search tree.
     */
    public TreeNode insertNode(TreeNode root, TreeNode node) {
        if (root == null) {
            return node;
        }
        if (root.val > node.val) {
            root.left = insertNode(root.left, node);
        } else {
            root.right = insertNode(root.right, node);
        }
        return root;
    }
}
```

Non-Recursion:

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
```

```
*         this.left = this.right = null;
*     }
* }
*/
public class Solution {
    /**
     * @param root: The root of the binary search tree.
     * @param node: insert this node into the binary search tree
     * @return: The root of the new binary search tree.
     */
    public TreeNode insertNode(TreeNode root, TreeNode node) {
        if (root == null) {
            root = node;
            return root;
        }
        TreeNode tmp = root;
        TreeNode last = null;
        while (tmp != null) {
            last = tmp;
            if (tmp.val > node.val) {
                tmp = tmp.left;
            } else {
                tmp = tmp.right;
            }
        }
        if (last != null) {
            if (last.val > node.val) {
                last.left = node;
            } else {
                last.right = node;
            }
        }
        return root;
    }
}
```

# Search Range in Binary Search Tree 11

## Question

Given two values  $k_1$  and  $k_2$  (where  $k_1 < k_2$ ) and a root pointer to a Binary Search Tree. Find all the keys of tree in range  $k_1$  to  $k_2$ . i.e. print all  $x$  such that  $k_1 \leq x \leq k_2$  and  $x$  is a key of given BST. Return all the keys in ascending order.

Example

If  $k_1 = 10$  and  $k_2 = 22$ , then your function should return  $[12, 20, 22]$ .

```
20
```

```
/\ 8 22 /\ 4 12
```

## Solution

D & C。若root值比 $k_1$ 小，则只搜root右子树，若root值比 $k_2$ 大，则只搜root左子树，否则左右子树都搜，然后先加左子树返回的答案，再加root，最后加右子树返回的答案。

代码如下：

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
```

```
/**
 * @param root: The root of the binary search tree.
 * @param k1 and k2: range k1 to k2.
 * @return: Return all keys that k1<=key<=k2 in ascending or
der.
 */
public ArrayList<Integer> searchRange(TreeNode root, int k1,
int k2) {
    // write your code here
    ArrayList<Integer> result = new ArrayList<Integer>();
    if(root == null){
        return result;
    }

    if(root.val < k1){
        return searchRange(root.right, k1, k2);
    }

    if(root.val > k2){
        return searchRange(root.left, k1, k2);
    }

    if(root.val >= k1 && root.val <= k2){
        ArrayList<Integer> left = searchRange(root.left, k1,
k2);
        ArrayList<Integer> right = searchRange(root.right, k
1, k2);

        result.addAll(left);
        result.add(root.val);
        result.addAll(right);
    }

    return result;
}
}
```

# Binary Search Tree Iterator 86

## Question

Design an iterator over a binary search tree with the following rules:

Elements are visited in ascending order (i.e. an in-order traversal) `next()` and `hasNext()` queries run in  $O(1)$  time in average.

Example

For the following binary search tree, in-order traversal by using iterator is [1, 6, 10, 11, 12]

```
10 /\ 1 11 \ \ 6 12
```

## Challenge

Extra memory usage  $O(h)$ ,  $h$  is the height of the tree.

Super Star: Extra memory usage  $O(1)$

## Solution

利用一个stack实现。

`next()`: 因为时in-order，所以要讲当前root所有左孩子入栈，直到最左边为止。将最左边元素出栈，因为左边必然已经没有节点，所以将当前节点设为出栈元素的右孩子，再次重复前一步操作。

`hasNext()`: 当前root不为null（证明还有左边节点未入栈）或者栈不为空（还有元素未出栈）时为true。

代码如下：

```
public class BSTIterator {
    private Stack<TreeNode> stack = new Stack<>();
    private TreeNode curt;

    // @param root: The root of binary tree.
    public BSTIterator(TreeNode root) {
        curt = root;
    }

    //@return: True if there has next node, or false
    public boolean hasNext() {
        return (curt != null || !stack.isEmpty());
    }

    //@return: return next node
    public TreeNode next() {
        while (curt != null) {
            stack.push(curt);
            curt = curt.left;
        }

        curt = stack.pop();
        TreeNode node = curt;
        curt = curt.right;

        return node;
    }
}
```

# Remove Node in Binary Search Tree 87

## Question

Given a root of Binary Search Tree with unique value for each node. Remove the node with given value. If there is no such a node with given value in the binary search tree, do nothing. You should keep the tree still a binary search tree after removal.

Example

Given binary search tree:

```
5
```

```
/ \ 3 6 / \ 2 4
```

Remove 3, you can either return:

```
5
```

```
/ \ 2 6 \ 4
```

or

```
5
```

```
/ \ 4 6 / 2
```

## Solution

首先在bst中搜索node值为value的点及其父节点。

下一步将该点从bst中删除。

1) 若要删除节点的右子树为空，则直接将要删除节点的左子树与其父节点连接。

2) 若要删除节点的右子树不为空，则找到右子树的最小值（右子树的最左边节点）使其成为以要删去节点为根节点的子树的新的根节点。

3) 首先将右子树最小值节点的右子树与右子树最小值节点的父节点连接（即从右子树中删去最小值节点，最小值节点在之前被保存），然后将要删去节点的父节点与最小值节点连接，最后将要删去节点的左右子树分别与最小值节点连接。

代码如下：

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param root: The root of the binary search tree.
     * @param value: Remove the node with given value.
     * @return: The root of the binary search tree after removal
     */
    public TreeNode removeNode(TreeNode root, int value) {
        // write your code here
        TreeNode dummy = new TreeNode(0);
        dummy.left = root;

        TreeNode parent = findNode(dummy, root, value);
        TreeNode node;
        if(parent.left != null && parent.left.val == value){
            node = parent.left;
        }else if(parent.right != null && parent.right.val == value){
            node = parent.right;
        }else{
            return root;
        }
        // write your code here
    }

    private TreeNode findNode(TreeNode dummy, TreeNode root, int value) {
        if(root == null) return null;
        if(root.val == value) return root;
        if(root.val < value) return findNode(dummy, root.right, value);
        return findNode(dummy, root.left, value);
    }
}
```



```
        //没找到
        return dummy.left;
    }

    deleteNode(parent, node);
    return dummy.left;
}

private TreeNode findNode(TreeNode parent, TreeNode node, int value){
    if(node == null){
        return parent;
    }

    if(node.val == value){
        return parent;
    }

    if(value < node.val){
        return findNode(node, node.left, value);
    }else{
        return findNode(node, node.right, value);
    }
}

private void deleteNode(TreeNode parent, TreeNode node){
    if(node.right == null){
        if(parent.left == node){
            parent.left = node.left;
        }else{
            parent.right = node.left;
        }
    }else{
        TreeNode temp = node.right;
        TreeNode father = node;

        while(temp.left != null){
            father = temp;
            temp = temp.left;
        }
    }
}
```

```
        if(father.left == temp){
            father.left = temp.right;
        }else{
            father.right = temp.right;
        }

        if(parent.left == node){
            parent.left = temp;
        }else{
            parent.right = temp;
        }

        temp.left = node.left;
        temp.right = node.right;
    }
}
```

# Inorder Successor in BST

## Question

Given a binary search tree and a node in it, find the in-order successor of that node in the BST.

Example

Given tree = [2,1] and node = 1:

2 / 1

return node 2.

Given tree = [2,1,3] and node = 2:

2 / \ 1 3

return node 3.

Note

If the given node has no in-order successor in the tree, return null.

Challenge

$O(h)$ , where  $h$  is the height of the BST.

## Solution

首先问面试官每个node是否有parent属性。

1) 首先要在bst中找到该节点。

2) 当该节点的右子树为空时，则向上寻找其父节点，父节点中第一个比该节点大的即为其后继节点。如果有parent属性则很好找，但是如果没有parent属性，则需要提前记住这个比该节点大的父节点。方法就是，每当root往左移动之前记录下这个root，向右移动则不需要，这样就能记录下与它最近的父节点。（再仔细说一下这个作用：纪录向左移动前的点就是纪录比root点大的下一个点，向右不需要纪

录是因为这个点本身比上一个点大，不需要纪录，这个点不可能成为右儿子的 successor)。这里还有一个技巧是记录的父节点初始化为null，这样就算要找的节点为最后一个节点（必然一直向右走，所以父节点一直为null），最后也可以返回父节点null。

3) 如果该节点右子树不为空，则其后继节点为右子树上最小值，因此只要找其右子树上最左边的节点即可。

代码如下：

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode inorderSuccessor(TreeNode root, TreeNode p)
    {
        TreeNode successor = null;
        while (root != null && root.val != p.val) {
            if (root.val > p.val) {
                successor = root;
                root = root.left;
            } else {
                root = root.right;
            }
        }

        if (root == null) {
            return null;
        }

        if (root.right == null) {
            return successor;
        }

        root = root.right;
        while (root.left != null) {
            root = root.left;
        }

        return root;
    }
}
```



# Segment Tree

区间树

[https://en.wikipedia.org/wiki/Segment\\_tree](https://en.wikipedia.org/wiki/Segment_tree)

[http://www.cnblogs.com/tanky\\_woo/archive/2010/09/25/1834523.html](http://www.cnblogs.com/tanky_woo/archive/2010/09/25/1834523.html)

# Segment Tree Build 201

## Question

The structure of Segment Tree is a binary tree which each node has two attributes start and end denote an segment / interval.

start and end are both integers, they should be assigned in following rules:

The root's start and end is given by build method.

The left child of node A has  $\text{start}=\text{A.left}$ ,  $\text{end}=(\text{A.left} + \text{A.right}) / 2$ .

The right child of node A has  $\text{start}=(\text{A.left} + \text{A.right}) / 2 + 1$ ,  $\text{end}=\text{A.right}$ .

if start equals to end, there will be no children for this node.

Implement a build method with two parameters start and end, so that we can create a corresponding segment tree with every node has the correct start and end value, return the root of this segment tree.

Clarification Segment Tree (a.k.a Interval Tree) is an advanced data structure which can support queries like:

which of these intervals contain a given point

which of these points are in a given interval

See wiki:

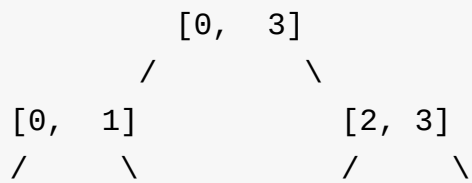
Segment Tree

Interval Tree

Example

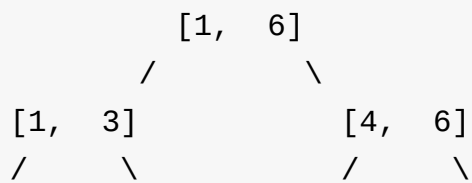
Given  $\text{start}=0$ ,  $\text{end}=3$ . The segment tree will be:





[0, 0] [1, 1] [2, 2] [3, 3]

Given start=1, end=6. The segment tree will be:



[1, 2] [3,3] [4, 5] [6,6] / \ / \ [1,1] [2,2] [4,4] [5,5]

## Solution

用d&c的方法建立左右子树后连起来即可。

代码如下：

```
/**
 * Definition of SegmentTreeNode:
 * public class SegmentTreeNode {
 *     public int start, end;
 *     public SegmentTreeNode left, right;
 *     public SegmentTreeNode(int start, int end) {
 *         this.start = start, this.end = end;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param start, end: Denote an segment / interval
     * @return: The root of Segment Tree
     */
    public SegmentTreeNode build(int start, int end) {
        // write your code here
        if(start > end){
            return null;
        }

        if(start == end){
            return new SegmentTreeNode(start, end);
        }

        int mid = (start + end) / 2;
        SegmentTreeNode root = new SegmentTreeNode(start, end);
        SegmentTreeNode left = build(start, mid);
        SegmentTreeNode right = build(mid + 1, end);

        root.left = left;
        root.right = right;

        return root;
    }
}
```



## Segment Tree Build II 439

### Question

The structure of Segment Tree is a binary tree which each node has two attributes start and end denote an segment / interval.

start and end are both integers, they should be assigned in following rules:

The root's start and end is given by build method.

The left child of node A has  $\text{start}=\text{A.left}$ ,  $\text{end}=(\text{A.left} + \text{A.right}) / 2$ .

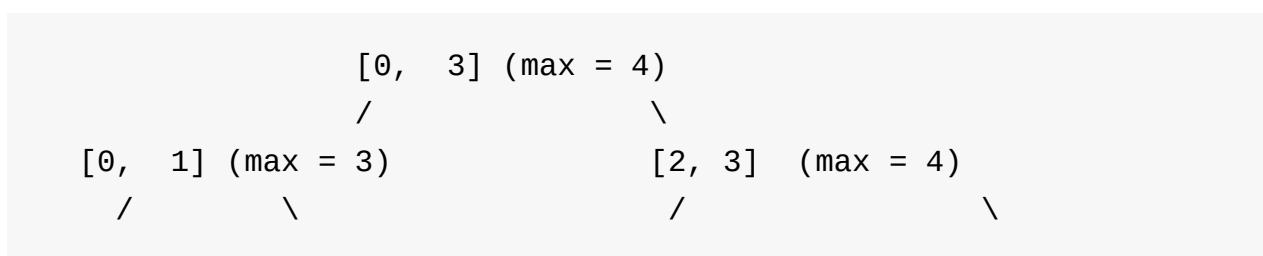
The right child of node A has  $\text{start}=(\text{A.left} + \text{A.right}) / 2 + 1$ ,  $\text{end}=\text{A.right}$ .

if start equals to end, there will be no children for this node.

Implement a build method with a given array, so that we can create a corresponding segment tree with every node value represent the corresponding interval max value in the array, return the root of this segment tree.

Example

Given [3,2,1,4]. The segment tree will be:



[0, 0] (max = 3) [1, 1] (max = 2) [2, 2] (max = 1) [3, 3] (max = 4)

### Solution

这题和I没有任何区别，只是多带一个属性max而已。

代码如下：

```
/**
 * Definition of SegmentTreeNode:
 * public class SegmentTreeNode {
 *     public int start, end, max;
 *     public SegmentTreeNode left, right;
 *     public SegmentTreeNode(int start, int end, int max) {
 *         this.start = start;
 *         this.end = end;
 *         this.max = max;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param A: a list of integer
     * @return: The root of Segment Tree
     */
    public SegmentTreeNode build(int[] A) {
        // write your code here
        if(A == null || A.length == 0){
            return null;
        }

        return buildHelper(A, 0, A.length - 1);
    }

    private SegmentTreeNode buildHelper(int[] A, int start, int
end){
        if(start == end){
            return new SegmentTreeNode(start, end, A[start]);
        }

        int mid = (start + end) / 2;
        SegmentTreeNode root = new SegmentTreeNode(start, end, 0
);
        SegmentTreeNode left = buildHelper(A, start, mid);
        SegmentTreeNode right = buildHelper(A, mid + 1, end);
    }
}
```

```
    root.left = left;
    root.right = right;
    root.max = Math.max(left.max, right.max);

    return root;
}
```

# Segment Tree Query 202

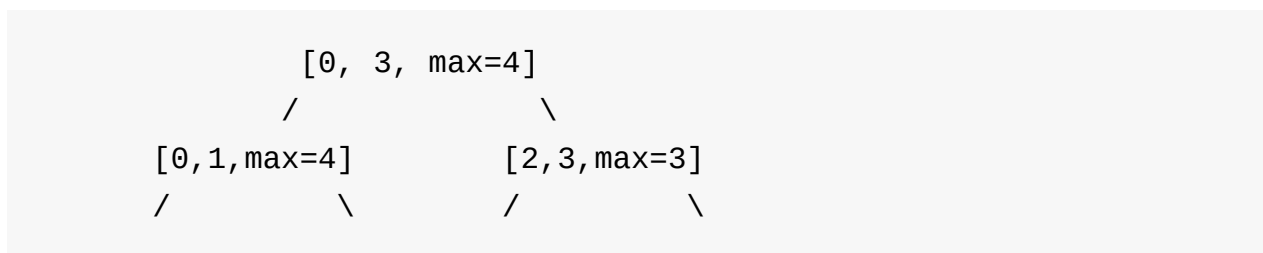
## Question

For an integer array (index from 0 to  $n-1$ , where  $n$  is the size of this array), in the corresponding SegmentTree, each node stores an extra attribute `max` to denote the maximum number in the interval of the array (index from `start` to `end`).

Design a query method with three parameters `root`, `start` and `end`, find the maximum number in the interval `[start, end]` by the given root of segment tree.

Example

For array `[1, 4, 2, 3]`, the corresponding Segment Tree is:



`[0,0,max=1] [1,1,max=4] [2,2,max=2], [3,3,max=3]`

`query(root, 1, 1)`, return 4

`query(root, 1, 2)`, return 4

`query(root, 2, 3)`, return 3

`query(root, 0, 2)`, return 4

## Solution

分情况讨论，递归查询：

1. 查询区间跨越`mid`，则要分别查询左边区间和右边区间，再取最大值
2. 查询区间只在一边，则只要查询那一边即可

代码如下：

```
/**
 * Definition of SegmentTreeNode:
 * public class SegmentTreeNode {
 *     public int start, end, max;
 *     public SegmentTreeNode left, right;
 *     public SegmentTreeNode(int start, int end, int max) {
 *         this.start = start;
 *         this.end = end;
 *         this.max = max;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param root, start, end: The root of segment tree and
     *                          an segment / interval
     * @return: The maximum number in the interval [start, end]
     */
    public int query(SegmentTreeNode root, int start, int end) {
        // write your code here
        if(root == null || start > end || start > root.end || end < root.start){
            return -1;
        }

        if((start == root.start && end == root.end) || (root.start == root.end)){
            return root.max;
        }

        int mid = (root.start + root.end) / 2;
        //跨越中点
        if(start <= mid && end >= mid + 1){
            int left = query(root.left, start, mid);
            int right = query(root.right, mid + 1, end);
            return Math.max(left, right);
        }
    }
}
```



```
    //左边
    if(end <= mid){
        return query(root.left, start, end);
    }
    //右边
    return query(root.right, start, end);
}
}
```

# Segment Tree Query II 247

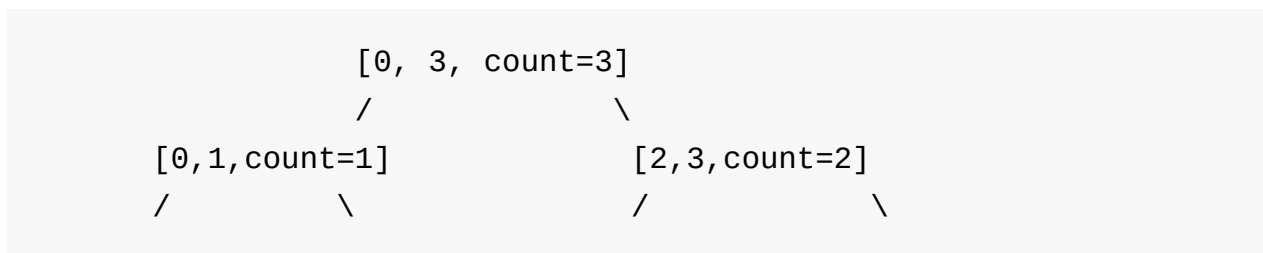
## Question

For an array, we can build a SegmentTree for it, each node stores an extra attribute count to denote the number of elements in the the array which value is between interval start and end. (The array may not fully filled by elements)

Design a query method with three parameters root, start and end, find the number of elements in the in array's interval [start, end] by the given root of value SegmentTree.

Example

For array [0, 2, 3], the corresponding value Segment Tree is:



[0,0,count=1] [1,1,count=0] [2,2,count=1], [3,3,count=1]

query(1, 1), return 0

query(1, 2), return 1

query(2, 3), return 2

query(0, 2), return 2

## Solution

和I一样，分情况讨论，递归解决：

1. 查询区间跨越mid，则要分别查询左边区间和右边区间，再将两边结果相加
2. 查询区间只在一边，则只要查询那一边即可

代码如下：

```
/**
 * Definition of SegmentTreeNode:
 * public class SegmentTreeNode {
 *     public int start, end, count;
 *     public SegmentTreeNode left, right;
 *     public SegmentTreeNode(int start, int end, int count) {
 *         this.start = start;
 *         this.end = end;
 *         this.count = count;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param root, start, end: The root of segment tree and
     *                          an segment / interval
     * @return: The count number in the interval [start, end]
     */
    public int query(SegmentTreeNode root, int start, int end) {
        // write your code here
        if(root == null || start > end || start > root.end || end < root.start){
            return 0;
        }

        if((start == root.start && end == root.end) || (root.start == root.end)){
            return root.count;
        }

        int mid = (root.start + root.end) / 2;
        if(start <= mid && end >= mid + 1){
            int left = query(root.left, start, mid);
            int right = query(root.right, mid + 1, end);
            return left + right;
        }else if(end <= mid){
            return query(root.left, start, end);
        }
    }
}
```

```
        }else if(start >= mid + 1){
            return query(root.right, start, end);
        }

        return 0;
    }
}
```

# Segment Tree Modify 203

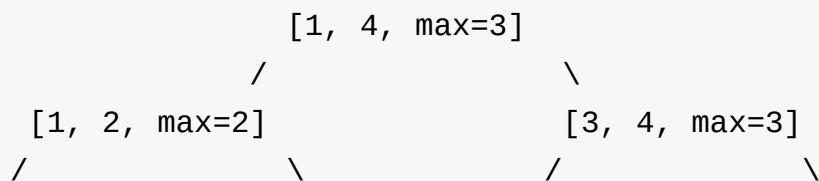
## Question

For a Maximum Segment Tree, which each node has an extra value max to store the maximum value in this node's interval.

Implement a modify function with three parameter root, index and value to change the node's value with [start, end] = [index, index] to the new given value. Make sure after this change, every node in segment tree still has the max attribute with the correct value.

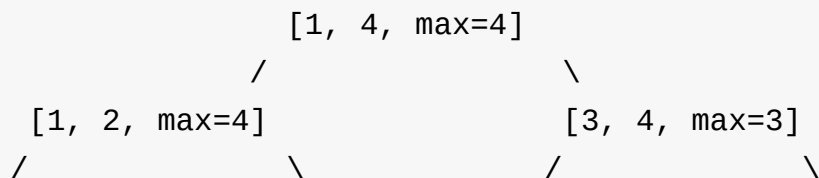
Example

For segment tree:



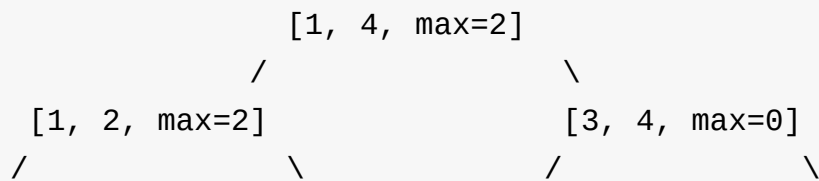
[1, 1, max=2], [2, 2, max=1], [3, 3, max=0], [4, 4, max=3]

if call modify(root, 2, 4), we can get:



[1, 1, max=2], [2, 2, max=4], [3, 3, max=0], [4, 4, max=3]

or call modify(root, 4, 0), we can get:



[1, 1, max=2], [2, 2, max=1], [3, 3, max=0], [4, 4, max=0]

### Challenge

Do it in  $O(h)$  time,  $h$  is the height of the segment tree.

## Solution

递归解决。一开始居然想到用stack记录整条查找路径再反向更新，真是蠢哭了。

1. index比mid小，在左边区间找
2. index比mid大，在右边区间找
3. 当某一点root的左右边界和index相等时，则找到，改变value

代码如下：

```

/**
 * Definition of SegmentTreeNode:
 * public class SegmentTreeNode {
 *     public int start, end, max;
 *     public SegmentTreeNode left, right;
 *     public SegmentTreeNode(int start, int end, int max) {
 *         this.start = start;
 *         this.end = end;
 *         this.max = max;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param root, index, value: The root of segment tree and
     * @param change the node's value with [index, index] to the new g

```

```
    given value
    /**@return: void
    */
    public void modify(SegmentTreeNode root, int index, int value) {
        // write your code here
        if(root == null || index < root.start || index > root.end){
            return;
        }

        if(root.start == index && root.end == index){
            root.max = value;
            return;
        }

        int mid = (root.start + root.end) / 2;
        if(index <= mid){
            modify(root.left, index, value);
        }else{
            modify(root.right, index, value);
        }

        root.max = Math.max(root.left.max, root.right.max);
    }
}
```

# Interval Minimum Number 205

## Question

Given an integer array (index from 0 to  $n-1$ , where  $n$  is the size of this array), and an query list. Each query has two integers  $[start, end]$ . For each query, calculate the minimum number between index  $start$  and  $end$  in the given array, return the result list.

Example

For array  $[1,2,7,8,5]$ , and queries  $[(1,2),(0,4),(2,4)]$ , return  $[2,1,5]$

Challenge

$O(\log N)$  time for each query

## Solution

水题，就是将build tree和query minimum结合起来求一连串的query而已。

代码如下：

```
/**
 * Definition of Interval:
 * public classs Interval {
 *     int start, end;
 *     Interval(int start, int end) {
 *         this.start = start;
 *         this.end = end;
 *     }
 */
class SegmentTreeNode{
    int start;
    int end;
    int min;
    SegmentTreeNode left;
```



```

        SegmentTreeNode right;
        public SegmentTreeNode(int start, int end, int min){
            this.start = start;
            this.end = end;
            this.min = min;
            left = right = null;
        }
    }

    public class Solution {
        /**
         * @param A, queries: Given an integer array and an query list
         * @return: The result list
         */
        public ArrayList<Integer> intervalMinNumber(int[] A,
                                                    ArrayList<Interval> queries) {
            // write your code here
            ArrayList<Integer> res = new ArrayList<Integer>();
            if(A == null || A.length == 0 || queries == null || queries.size() == 0){
                return res;
            }

            SegmentTreeNode root = SegmentTreeBuilder(A, 0, A.length - 1);

            for(int i = 0; i < queries.size(); i++){
                int min = query(root, queries.get(i).start, queries.get(i).end);
                res.add(min);
            }

            return res;
        }

        private SegmentTreeNode SegmentTreeBuilder(int[] A, int start, int end){
            if(start == end){

```

```
        return new SegmentTreeNode(start, end, A[start]);
    }

    int mid = (start + end) / 2;
    SegmentTreeNode root = new SegmentTreeNode(start, end, 0
);
    SegmentTreeNode left = SegmentTreeBuilder(A, start, mid)
;
    SegmentTreeNode right = SegmentTreeBuilder(A, mid + 1, e
nd);

    root.left = left;
    root.right = right;
    root.min = Math.min(left.min, right.min);

    return root;
}

public int query(SegmentTreeNode root, int start, int end) {
    // write your code here
    if(root == null || start > end || start > root.end || en
d < root.start){
        return -1;
    }

    if(start == root.start && end == root.end){
        return root.min;
    }

    int mid = (root.start + root.end) / 2;
    //跨越中点
    if(start <= mid && end >= mid + 1){
        int left = query(root.left, start, mid);
        int right = query(root.right, mid + 1, end);
        return Math.min(left, right);
    }
    //左边
    if(end <= mid){
        return query(root.left, start, end);
    }
}
```

```
        //右边
        return query(root.right, start, end);
    }
}
```

# Interval Sum 206

## Question

Given an integer array (index from 0 to  $n-1$ , where  $n$  is the size of this array), and an query list. Each query has two integers  $[start, end]$ . For each query, calculate the sum number between index start and end in the given array, return the result list.

Example

For array  $[1,2,7,8,5]$ , and queries  $[(0,4),(1,2),(2,4)]$ , return  $[23,9,20]$

Challenge

$O(\log N)$  time for each query

## Solution

水题，就是将build tree和query sum结合起来而已。

代码如下：

```
/**
 * Definition of Interval:
 * public classs Interval {
 *     int start, end;
 *     Interval(int start, int end) {
 *         this.start = start;
 *         this.end = end;
 *     }
 */
class SegmentTreeNode{
    int start;
    int end;
    long sum;
    SegmentTreeNode left;
```

```

    SegmentTreeNode right;
    public SegmentTreeNode(int start, int end, long sum){
        this.start = start;
        this.end = end;
        this.sum = sum;
        left = right = null;
    }
}

public class Solution {
    /**
     * @param A, queries: Given an integer array and an query list
     * @return: The result list
     */
    public ArrayList<Long> intervalSum(int[] A,
                                       ArrayList<Interval> queries) {
        // write your code here
        ArrayList<Long> res = new ArrayList<Long>();
        if(A == null || A.length == 0 || queries == null || queries.size() == 0){
            return res;
        }

        SegmentTreeNode root = SegmentTreeBuilder(A, 0, A.length - 1);

        for(int i = 0; i < queries.size(); i++){
            long sum = query(root, queries.get(i).start, queries.get(i).end);
            res.add(sum);
        }

        return res;
    }

    private SegmentTreeNode SegmentTreeBuilder(int[] A, int start, int end){
        if(start == end){

```

```
        return new SegmentTreeNode(start, end, (long)A[start
]);
    }

    int mid = (start + end) / 2;
    SegmentTreeNode root = new SegmentTreeNode(start, end, 0
);
    SegmentTreeNode left = SegmentTreeBuilder(A, start, mid)
;
    SegmentTreeNode right = SegmentTreeBuilder(A, mid + 1, e
nd);

    root.left = left;
    root.right = right;
    root.sum = left.sum + right.sum;

    return root;
}

public long query(SegmentTreeNode root, int start, int end)
{
    // write your code here
    if(root == null || start > end || start > root.end || en
d < root.start){
        return -1;
    }

    if(start == root.start && end == root.end){
        return root.sum;
    }

    int mid = (root.start + root.end) / 2;
    //跨越中点
    if(start <= mid && end >= mid + 1){
        long left = query(root.left, start, mid);
        long right = query(root.right, mid + 1, end);
        return left + right;
    }
    //左边
    if(end <= mid){
```

```
        return query(root.left, start, end);
    }
    //右边
    return query(root.right, start, end);
}
}
```

# Interval Sum II 207

## Question

Given an integer array in the construct method, implement two methods query(start, end) and modify(index, value):

For query(start, end), return the sum from index start to index end in the given array. For modify(index, value), modify the number in the given index to value.

Example

Given array A = [1,2,7,8,5].

query(0, 2), return 10.

modify(0, 4), change A[0] from 1 to 4.

query(0, 1), return 6.

modify(2, 1), change A[2] from 7 to 1.

query(2, 4), return 14.

Challenge

O(logN) time for query and modify.

## Solution

水题，就是把query sum和modify结合起来而已。

代码如下：

```
public class Solution {  
    /* you may need to use some attributes here */  
    class SegmentTreeNode{  
        int start;  
        int end;
```



```
        long sum;
        SegmentTreeNode left;
        SegmentTreeNode right;
        public SegmentTreeNode(int start, int end, long sum){
            this.start = start;
            this.end = end;
            this.sum = sum;
            left = right = null;
        }
    }

    SegmentTreeNode root;

    /**
     * @param A: An integer array
     */
    public Solution(int[] A) {
        // write your code here
        if(A == null || A.length == 0){
            root = null;
        }else{
            root = SegmentTreeBuilder(A, 0, A.length - 1);
        }
    }

    private SegmentTreeNode SegmentTreeBuilder(int[] A, int start, int end){
        if(start == end){
            return new SegmentTreeNode(start, end, (long)A[start]);
        }

        int mid = (start + end) / 2;
        SegmentTreeNode root = new SegmentTreeNode(start, end, 0);
        SegmentTreeNode left = SegmentTreeBuilder(A, start, mid);
        SegmentTreeNode right = SegmentTreeBuilder(A, mid + 1, end);
    }
```

```

        root.left = left;
        root.right = right;
        root.sum = left.sum + right.sum;

        return root;
    }

    /**
     * @param start, end: Indices
     * @return: The sum from start to end
     */
    public long query(int start, int end) {
        // write your code here
        if(root == null || start > end || start > root.end || end < root.start){
            return -1;
        }

        return queryHelper(root, start, end);
    }

    public long queryHelper(SegmentTreeNode r, int start, int end){
        if((start == r.start && end == r.end) || (r.start == r.end)){
            return r.sum;
        }

        int mid = (r.start + r.end) / 2;
        //跨越中点
        if(start <= mid && end >= mid + 1){
            long left = queryHelper(r.left, start, mid);
            long right = queryHelper(r.right, mid + 1, end);
            return left + right;
        }
        //左边
        if(end <= mid){
            return queryHelper(r.left, start, end);
        }
        //右边

```

```
        return queryHelper(r.right, start, end);
    }

    /**
     * @param index, value: modify A[index] to value.
     */
    public void modify(int index, int value) {
        // write your code here
        if(root == null || index < root.start || index > root.end){
            return;
        }

        modifyHelper(root, index, value);
    }

    public void modifyHelper(SegmentTreeNode root, int index, int value) {
        // write your code here
        if(root.start == index && root.end == index){
            root.sum = value;
            return;
        }

        int mid = (root.start + root.end) / 2;
        if(index <= mid){
            modifyHelper(root.left, index, value);
        }else{
            modifyHelper(root.right, index, value);
        }

        root.sum = root.left.sum + root.right.sum;
    }
}
```

# Count of Smaller Number before itself 249

## Question

Give you an integer array (index from 0 to n-1, where n is the size of this array, value from 0 to 10000) . For each element  $A_i$  in the array, count the number of element before this element  $A_i$  is smaller than it and return count number array.

Example

For array [1,2,7,8,5], return [0,1,2,3,2]

## Solution

SegmentTree里唯一一道还有点难度的题。

1. 首先以0-10000为区间建树，并将所有区间count设为0。每一个最小区间（即叶节点）的count代表到目前为止遇到的该数的数量。
2. 然后开始遍历数组，遇到 $A[i]$ 时，去查0- $A[i]-1$ 区间的count，即为比 $A[i]$ 小的数的数量
3. 查完后将 $A[i]$ 区间的count加1即可

代码如下：

```
public class Solution {
    /**
     * @param A: An integer array
     * @return: Count the number of element before this element
     * 'ai' is
     *         smaller than it and return count number array
     */
    class SegmentTreeNode {
        public int start, end;
        public int count;
        public SegmentTreeNode left, right;
        public SegmentTreeNode(int start, int end, int count) {
```

```
        this.start = start;
        this.end = end;
        this.count = count;
        this.left = this.right = null;
    }
}

SegmentTreeNode root;

private SegmentTreeNode build(int start, int end){
    if(start > end){
        return null;
    }

    SegmentTreeNode root = new SegmentTreeNode(start, end, 0
);

    if(start != end){
        int mid = (start + end) / 2;
        root.left = build(start, mid);
        root.right = build(mid + 1, end);
    }

    return root;
}

private int query(SegmentTreeNode root, int start, int end){
    if(start == root.start && end == root.end){
        return root.count;
    }

    int leftCount = 0;
    int rightCount = 0;

    int mid = (root.start + root.end) / 2;
    if(start <= mid){
        if(end > mid){
            leftCount = query(root.left, start, mid);
        }else{
            leftCount = query(root.left, start, end);
        }
    }
}
```

```
        }
    }

    if(end > mid){
        if(start <= mid){
            rightCount = query(root.right, mid + 1, end);
        }else{
            rightCount = query(root.right, start, end);
        }
    }

    return leftCount + rightCount;
}

private void modify(SegmentTreeNode root, int index, int value){
    if(root.start == index && root.end == index){
        root.count += value;
        return;
    }

    int mid = (root.start + root.end) / 2;
    if(index >= root.start && index <= mid){
        modify(root.left, index, value);
    }

    if(index <= root.end && index > mid){
        modify(root.right, index, value);
    }

    root.count = root.left.count + root.right.count;
}

public ArrayList<Integer> countOfSmallerNumberII(int[] A) {
    // write your code here
    ArrayList<Integer> result = new ArrayList<Integer>();
    if(A == null || A.length == 0){
        return result;
    }
}
```

```
    root = build(0, 10000);
    int ans;
    for(int i = 0; i < A.length; i++){
        ans = 0;
        if(A[i] > 0){
            ans = query(root, 0, A[i] - 1);
        }
        modify(root, A[i], 1);
        result.add(ans);
    }

    return result;
}
```

# Linear



# Array

## Median 80

### Question

Given a unsorted array with integers, find the median of it.

A median is the middle number of the array after it is sorted.

If there are even numbers in the array, return the  $N/2$ -th number after sorted.

Example

Given [4, 5, 1, 2, 3], return 3.

Given [7, 9, 4, 5], return 5.

Challenge

$O(n)$  time.

### Solution

用quicksort（或者其它排序法）排序后再找median。

没有找到 $O(n)$ 的方法。

代码如下：

```
public class Solution {  
    /**  
     * @param nums: A list of integers.  
     * @return: An integer denotes the middle number of the array.  
     */  
    public int median(int[] nums) {  
        // write your code here  
        quickSort(nums, 0, nums.length - 1);  
  
        if(nums.length % 2 == 0){
```

```
        return nums[nums.length / 2 - 1];
    }

    return nums[nums.length / 2];
}

private void quickSort(int[] nums, int start, int end){
    if(start < end){
        int q = partition(nums, start, end);
        quickSort(nums, start, q - 1);
        quickSort(nums, q + 1, end);
    }
}

private int partition(int[] nums, int start, int end){
    int pivot = nums[end];
    int i = start - 1;
    for(int j = start; j < end; j++){
        if(nums[j] < pivot){
            i += 1;
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
        }
    }
    int temp = nums[i + 1];
    nums[i + 1] = nums[end];
    nums[end] = temp;

    return i + 1;
}
}
```

# Remove Duplicates from Sorted Array 100

## Question

Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

Example

Given input array A = [1,1,2],

Your function should return length = 2, and A is now [1,2].

## Solution

idea：从数组开头到nums[i]的所有数均为不重复的数。

i从0开始（第一个数），若遇到一个和第一个数不一样的数（即第二个数），则将i进一位并将nums[i]赋值为第二个数，依次类推，直到数组最后一个数。因此，共有i+1个数是不重复的数。

代码如下：

```
public class Solution {  
    /**  
     * @param A: a array of integers  
     * @return : return an integer  
     */  
    public int removeDuplicates(int[] nums) {  
        // write your code here  
        if(nums == null || nums.length == 0){  
            return 0;  
        }  
  
        int i = 0;  
        for(int j = 1; j < nums.length; j++){  
            if(nums[j] != nums[i]){  
                i++;  
                nums[i] = nums[j];  
            }  
        }  
  
        return i + 1;  
    }  
}
```

## Follwup Question

Follow up for "Remove Duplicates":

What if duplicates are allowed at most twice?

For example,

Given sorted array A = [1,1,1,2,2,3],

Your function should return length = 5, and A is now [1,1,2,2,3].

## Solution

额外用一个boolean表示是否可以添加重复元素，只有boolean为true时才能添加重复元素。当添加一个新元素时，boolean设置为true，当添加一个重复元素后，boolean设置为false。

代码如下：

```
public class Solution {
    /**
     * @param A: a array of integers
     * @return : return an integer
     */
    public int removeDuplicates(int[] nums) {
        // write your code here
        if(nums == null || nums.length == 0){
            return 0;
        }

        boolean twice = true;
        int i = 0;
        for(int j = 1; j < nums.length; j++){
            if(nums[j] != nums[i]){
                i++;
                nums[i] = nums[j];
                twice = true;
            }else if(twice){
                i++;
                nums[i] = nums[j];
                twice = false;
            }
        }
        return i + 1;
    }
}
```

# Merge Intervals 156

## Question

Given a collection of intervals, merge all overlapping intervals.

Example

Given intervals => merged intervals:

```
[           [
  [1, 3],      [1, 6],
  [2, 6],      =>  [8, 10],
  [8, 10],     [15, 18]
  [15, 18]     ]
]
```

Challenge

$O(n \log n)$  time and  $O(1)$  extra space.

## Solution

先对list中的interval以start排序，然后依次合并有交集的interval。需要重构一个 comparator。

代码如下：

```
/**
 * Definition of Interval:
 * public class Interval {
 *     int start, end;
 *     Interval(int start, int end) {
 *         this.start = start;
 *         this.end = end;
 *     }
 */
```

```
class Solution {
    /**
     * @param intervals, a collection of intervals
     * @return: A new sorted interval list.
     */
    public List<Interval> merge(List<Interval> intervals) {
        // write your code here
        Collections.sort(intervals, new Comparator<Interval>(){
            public int compare(Interval a, Interval b){
                return a.start - b.start;
            }
        });

        for(int i = 0; i < intervals.size() - 1; i++){
            if(intervals.get(i + 1).start <= intervals.get(i).end){
                if(intervals.get(i + 1).end > intervals.get(i).end){
                    intervals.get(i).end = intervals.get(i + 1).end;
                }else{
                    intervals.remove(i + 1);
                }
                i--;
            }
        }

        return intervals;
    }
}
```



# First Missing Positive 189

## Question

## Solution

Idea: 数组第 $x$ 位上的数字应该是 $x+1$ 。遍历数组，将每一个数送到它应该在的位置上。然后再次遍历新数组，找到第一个第 $x$ 位上不等于 $x+1$ 的数。将数组元素调整到其应该在的位置上的具体操作如下：

- 1) 遍历数组，在第 $i$ 位时，若为正数且数值小于 $A.length$ 且第 $i$ 位上的数不等于 $i+1$ 时，可以将该位上的数调整到其应该在的位置。根据第 $x$ 位的数应该为 $x+1$ 的规则，则该位上的数应该在的位置为 $A[i]-1$ 。这里详细说一下，若该位为负数或数值大于 $A.length$ 则得到其应该在的位置，所以无法调整，继续下一个数。
- 2) 看其应该在的位置上（ $A[i]-1$ ）的数是否为正确的数（ $A[i]$ ），若已经正确，则无需调整，否则将第 $i$ 位数与第 $A[i]-1$ 位数进行互换。对新换到 $i$ 位上的数重复上述步骤。（因此要用while）
- 3) 遍历整个数组则所有能调整的数都已经被调整到其合适的位置上
- 4) 再次遍历数组，找到第一个第 $x$ 位上不等于 $x+1$ 的数并返回 $x+1$

代码如下：

```
public class Solution {
    /**
     * @param A: an array of integers
     * @return: an integer
     */
    public int firstMissingPositive(int[] A) {
        // write your code here
        if(A == null || A.length == 0){
            return 1;
        }

        for(int i = 0; i < A.length; i++){
            while(A[i] > 0 && A[i] != (i + 1) && A[i] <= A.length
h){
                int index = A[i] - 1;
                if(A[index] == A[i]){
                    break;
                }
                int temp = A[i];
                A[i] = A[index];
                A[index] = temp;
            }
        }

        for(int i = 0; i < A.length; i++){
            if(A[i] != i + 1){
                return i + 1;
            }
        }

        return A.length + 1;
    }
}
```

# Partition Array by Odd and Even 373

## Question

Partition an integers array into odd number first and even number second.

Example

Given [1, 2, 3, 4], return [1, 3, 2, 4]

Challenge

Do it in-place.

## Solution

前后指针。前指针从前往后遇到偶数停下，后指针从后往前遇到奇数停下，交换，继续，直到前后指针交叠。注意的地方是前指针从前往后走的时候要注意不要越界，后指针也一样。

代码如下：

```
public class Solution {
    /**
     * @param nums: an array of integers
     * @return: nothing
     */
    public void partitionArray(int[] nums) {
        // write your code here;
        for(int i = 0, j = nums.length - 1; i < j; i++, j--){
            while(i < nums.length && nums[i] % 2 != 0){
                i++;
            }
            while(j >= 0 && nums[j] % 2 == 0){
                j--;
            }
            if(i < j){
                int temp = nums[i];
                nums[i] = nums[j];
                nums[j] = temp;
            }
        }
    }
}
```

# Trapping Rain Water 363

## Question

Given  $n$  non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.



### Example

Given `[0,1,0,2,1,0,1,3,2,1,2,1]`, return 6.

### Challenge

$O(n)$  time and  $O(1)$  memory

$O(n)$  time and  $O(n)$  memory is also acceptable.

## Solution

前后指针法，时间复杂度 $O(n)$ ，空间复杂度 $O(1)$ 。

1) 前指针指向数组头，后指针指向数组尾。

2) 比较前后指针，从小的那个开始（记录此时最初指针的值smaller），

3) 向后或前移动指针，当遇到小于等于它的元素(index为*i*)时，新增加的面积为 `smaller-heights[i]`，当遇到大于它的元素时停止.重复2-3步骤，直到前后指针交叠。

此处要说明的是，在第2步移动的必须是较小值的指针，这就好比一根短木板和一根长木板，若以长木板为基准向短木板推进，水会漏出，反之则可以。

代码如下：

```
public class Solution {
    /**
     * @param heights: an array of integers
     * @return: a integer
     */
    public int trapRainWater(int[] heights) {
        // write your code here
        if(heights == null || heights.length == 0){
            return 0;
        }

        int start = 0;
        int end = heights.length - 1;
        int area = 0;
        while(start < end){
            if(heights[start] < heights[end]){
                int smaller = heights[start];
                while(start < end && heights[start] <= smaller){
                    area += smaller - heights[start];
                    start++;
                }
            }else{
                int smaller = heights[end];
                while(start < end && heights[end] <= smaller){
                    area += smaller - heights[end];
                    end--;
                }
            }
        }
        return area;
    }
}
```

## Move Zeroes 539

### Question

Given an array `nums`, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.

Notice

You must do this in-place without making a copy of the array.

Minimize the total number of operations.

Example

Given `nums = [0, 1, 0, 3, 12]`, after calling your function, `nums` should be `[1, 3, 12, 0, 0]`.

### Solution

双指针法two pointers。第一个指针遇到0就停下（在0之前的位置），第二个指针继续寻找之后第一个非0的元素，找到之后和第一个指针后面的0交换，第一个指针移动一位。即第一个指针和第二个指针之间的元素为0。

代码如下：

```
public class Solution {  
    /**  
     * @param nums an integer array  
     * @return nothing, do this in-place  
     */  
    public void moveZeroes(int[] nums) {  
        // Write your code here  
        if(nums == null || nums.length == 0){  
            return;  
        }  
  
        int i = -1;;  
        for(int j = 0; j < nums.length; j++){  
            if(nums[j] != 0){  
                i++;  
                int temp = nums[i];  
                nums[i] = nums[j];  
                nums[j] = temp;  
            }  
        }  
    }  
}
```



## Spiral Matrix 374

### Question

Given a matrix of  $m \times n$  elements ( $m$  rows,  $n$  columns), return all elements of the matrix in spiral order.

Example

Given the following matrix:

```
[[ 1, 2, 3 ],
```

```
 [ 4, 5, 6 ],
```

```
 [ 7, 8, 9 ]]
```

You should return `[1,2,3,6,9,8,7,4,5]`.

### Solution

一圈一圈（上->右->下->左）输出matrix。每一轮消耗2层row和2层column。边界情况为到达最内层时，若row或者column为偶数，则和之前一样输出；若row或者column为奇数，则最内层只剩下一行或者一列，只要输出此时的上->右即可。

代码如下：

Non-Recursion:

```
public class Solution {
    public ArrayList<Integer> spiralOrder(int[][] matrix) {
        ArrayList<Integer> rst = new ArrayList<Integer>();
        if(matrix == null || matrix.length == 0)
            return rst;

        int rows = matrix.length;
        int cols = matrix[0].length;
        int count = 0;
        while(count * 2 < rows && count * 2 < cols){
            for(int i = count; i < cols-count; i++){
                rst.add(matrix[count][i]);

                for(int i = count+1; i < rows-count; i++){
                    rst.add(matrix[i][cols-count-1]);

                    if(rows - 2 * count == 1 || cols - 2 * count == 1)
                        // if only one row /col remains
                        break;

                    for(int i = cols-count-2; i >= count; i--){
                        rst.add(matrix[rows-count-1][i]);

                        for(int i = rows-count-2; i >= count+1; i--){
                            rst.add(matrix[i][count]);

                            count++;
                        }
                    }
                }
            }
        }
        return rst;
    }
}
```

Recursion:

```
public class Solution {
    /**
     * @param matrix a matrix of m x n elements
     * @return an integer list
     */
}
```

```
    */
    public List<Integer> spiralOrder(int[][] matrix) {
        // Write your code here
        List<Integer> result = new ArrayList<Integer>();
        if(matrix == null || matrix.length == 0 || matrix[0].length == 0){
            return result;
        }

        int row = matrix.length;
        int col = matrix[0].length;

        Helper(matrix, 0, row, col, result);
        return result;
    }

    private void Helper(int[][] matrix, int distance, int row, int col, List<Integer> result){
        //row或column为偶数
        if(row - distance * 2 == 0 || col - distance * 2 == 0){
            return;
        }

        //只剩一行
        if(row - distance * 2 == 1){
            for(int j = distance; j <= col - 1 - distance; j++){
                result.add(matrix[distance][j]);
            }
            return;
        }
        //只剩一列
        if(col - distance * 2 == 1){
            for(int i = distance; i <= row - 1 - distance; i++){
                result.add(matrix[i][distance]);
            }
            return;
        }
        //上
        for(int j = distance; j <= col - 1 - distance; j++){
            result.add(matrix[distance][j]);
        }
    }
}
```

```
    }  
    //右  
    for(int i = distance + 1; i <= row - 2 - distance; i++){  
        result.add(matrix[i][col - 1 - distance]);  
    }  
    //下  
    for(int j = col - 1 - distance; j >= distance; j--){  
        result.add(matrix[row - 1 - distance][j]);  
    }  
    //左  
    for(int i = row - 2 - distance; i >= distance + 1; i--){  
        result.add(matrix[i][distance]);  
    }  
  
    Helper(matrix, distance + 1, row, col, result);  
}  
}
```

## Spiral Matrix II 381

### Question

Given an integer  $n$ , generate a square matrix filled with elements from 1 to  $n^2$  in spiral order.

Example

Given  $n = 3$ ,

You should return the following matrix:

```
[[ 1, 2, 3 ],
```

```
 [ 8, 9, 4 ],
```

```
 [ 7, 6, 5 ]]
```

### Solution

与Spiral Matrix一样，只不过这次是往里加数。

代码如下：

```
public class Solution {
    /**
     * @param n an integer
     * @return a square matrix
     */
    public int[][] generateMatrix(int n) {
        // Write your code here
        int[][] matrix = new int[n][n];

        int number = 1;
        int count = 0;
        while(count * 2 < n){
            for(int i = count; i < n - count; i++){
                matrix[count][i] = number++;
            }

            for(int i = count + 1; i < n - count; i++){
                matrix[i][n - 1 - count] = number++;
            }

            if(n - count * 2 == 1){
                break;
            }

            for(int i = n - 2 - count; i >= count; i--){
                matrix[n - 1 - count][i] = number++;
            }

            for(int i = n - 2 - count; i > count; i--){
                matrix[i][count] = number++;
            }
            count++;
        }
        return matrix;
    }
}
```



# Plus One 407

## Question

Given a non-negative number represented as an array of digits, plus one to the number.

The digits are stored such that the most significant digit is at the head of the list.

Example

Given [1,2,3] which represents 123, return [1,2,4].

Given [9,9,9] which represents 999, return [1,0,0,0].

## Solution

每一位取出之后加上之前一位的进位，%10之后为该位的数字，/10之后为该位的进位。有几点注意：

- 1) 若前一位的进位为0，可以直接停止循环
- 2) 若最左边一位的进位为0可以直接返回，否则要用一个长度加1的新数组的第一位保存1，其它各位与原数组相同
- 3) 时间复杂度是 $O(1)$

代码如下：



```
public class Solution {
    // The complexity is O(1)
    //  $f(n) = 9/10 + 1/10 * O(n-1)$ 
    //  $\Rightarrow O(n) = 10 / 9 = 1.1111 = O(1)$ 

    public int[] plusOne(int[] digits) {
        int carries = 1;
        for(int i = digits.length-1; i>=0 && carries > 0; i--){
            // fast break when carries equals zero
            int sum = digits[i] + carries;
            digits[i] = sum % 10;
            carries = sum / 10;
        }
        if(carries == 0)
            return digits;

        int[] rst = new int[digits.length+1];
        rst[0] = 1;
        for(int i=1; i< rst.length; i++){
            rst[i] = digits[i-1];
        }
        return rst;
    }
}
```

# Container With Most Water 383

## Question

Given  $n$  non-negative integers  $a_1, a_2, \dots, a_n$ , where each represents a point at coordinate  $(i, a_i)$ .  $n$  vertical lines are drawn such that the two endpoints of line  $i$  is at  $(i, a_i)$  and  $(i, 0)$ . Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Notice

You may not slant the container.

Example

Given  $[1,3,2]$ , the max area of the container is 2.

## Solution

这题和Trapping rain water不一样，取任意两个木板后，中间的木板可以忽略。

首先想到的是两次循环找最大值，时间复杂度 $O(n^2)$ 。

如果用前后指针则可以将时间复杂度降到 $O(n)$ 。计算取前后指针为木板时的面积，然后移动较短的那一个。保留最大面积。

代码如下：

$O(n^2)$

```
public class Solution {  
    /**  
     * @param heights: an array of integers  
     * @return: an integer  
     */  
    public int maxArea(int[] heights) {  
        // write your code here  
        if(heights == null || heights.length <= 1){  
            return 0;  
        }  
        int area = 0;  
        for(int i = 0; i < heights.length - 1; i++){  
            for(int j = i + 1; j < heights.length; j++){  
                int width = j - i;  
                int height = Math.min(heights[i], heights[j]);  
                area = Math.max(area, width * height);  
            }  
        }  
        return area;  
    }  
}
```

$O(n)$

```
public class Solution {
    /**
     * @param heights: an array of integers
     * @return: an integer
     */
    public int maxArea(int[] heights) {
        // write your code here
        if(heights == null || heights.length <= 1){
            return 0;
        }

        int start = 0;
        int end = heights.length - 1;
        int sum = 0;
        while(start < end){
            sum = Math.max(sum, (end - start) * Math.min(heights
[start], heights[end]));
            if(heights[start] < heights[end]){
                start++;
            }else{
                end--;
            }
        }
        return sum;
    }
}
```

# The Smallest Difference 387

## Question

Given two array of integers(the first array is array A, the second array is array B), now we are going to find a element in array A which is  $A[i]$ , and another element in array B which is  $B[j]$ , so that the difference between  $A[i]$  and  $B[j]$  ( $|A[i] - B[j]|$ ) is as small as possible, return their smallest difference.

### Example

For example, given array  $A = [3,6,7,4]$ ,  $B = [2,8,9,3]$ , return 0

### Challenge

$O(n \log n)$  time

## Solution

分别对两个数组排序，然后从两个数组最小的元素开始比较，更新最小值。因为两个数组是顺序排列，增大较小的那个数才有可能缩小两者的差值，因此每次将较小的那个数移动一位。当一个数组为空时，表明已经不可能再增大，此时的差值已经是能达到的最小。

时间复杂度 $O(n \log n)$ 。

代码如下：

```
public class Solution {
    /**
     * @param A, B: Two integer arrays.
     * @return: Their smallest difference.
     */
    public int smallestDifference(int[] A, int[] B) {
        // write your code here
        if(A == null || A.length == 0 || B == null || B.length =
= 0){
            return 0;
        }

        Arrays.sort(A);
        Arrays.sort(B);

        int a = 0; int b = 0;
        int min = Integer.MAX_VALUE;
        while(a < A.length && b < B.length){
            min = Math.min(min, Math.abs(A[a] - B[b]));
            if(A[a] < B[b]){
                a++;
            }else{
                b++;
            }
        }
        return min;
    }
}
```

# Permutation Sequence 388

## Question

Given  $n$  and  $k$ , return the  $k$ -th permutation sequence.

Notice

$n$  will be between 1 and 9 inclusive.

Example

For  $n = 3$ , all permutations are listed as follows:

"123"

"132"

"213"

"231"

"312"

"321"

If  $k = 4$ , the fourth permutation is "231"

Challenge

$O(n*k)$  in time complexity is easy, can you do it in  $O(n^2)$  or less?

## Solution

先将所有可用的数字（1- $n$ ）存入一个list，每用掉一个数就将其从list中删去。

idea: 以某一数字开头的排列有 $(n-1)!$ 个。例如：123，132，以1开头的是2！个。

1) 第一位数字可以用 $(k-1)/(n-1)!$ 来计算，相当于一轮 $(n-1)!$ 就换一个数，看有几轮就是第几个数。这里用 $k-1$ 是为了在边界条件时计算正确，例如 $k$ 正好等于 $(n-1)!$ ，则应该是第0轮的最后一个数，所以应该取第0个数，但如果直接用 $k/(n-1)!$ 得到的是1，所以一开始要用 $k-1$ ，这样边界时的计算才是正确的。

2) 剩下的 $k$ 为 $(k-1) \bmod (n-1)!$ 。

3) 重复第一步，用新的 $k$ 来计算第二位的数。此时每一轮的大小应该变为 $(n-2)!$ ，用 $k/(n-2)!$ 来得到在剩下可用的数中要取第几位数。

4) 重复1-3直到所有可用的数字用尽。

代码如下：



```
class Solution {
    /**
     * @param n: n
     * @param k: the kth permutation
     * @return: return the k-th permutation
     */
    public String getPermutation(int n, int k) {
        ArrayList<Character> digits = new ArrayList<Character>()
;
        for(char i = '1'; i < '1' + n; i++){
            digits.add(i);
        }

        int factor = 1;
        for(int i = 1; i <= n; i++){
            factor *= i;
        }

        StringBuilder sb = new StringBuilder();
        k = k - 1;
        int count = n;
        while(!digits.isEmpty()){
            factor /= count;
            count--;
            int index = k / factor;
            k = k % factor;
            sb.append(digits.get(index));
            digits.remove(index);
        }
        return sb.toString();
    }
}
```

# Number of Airplanes in the Sky 391

## Question

Given an interval list which are flying and landing time of the flight. How many airplanes are on the sky at most?

Notice

If landing and flying happens at the same time, we consider landing should happen at first.

Example

For interval list

[ [1,10],

[2,3],

[5,8],

[4,7] ]

Return 3

## Solution

- 1) 将start和end时间分别保存在两个list中，并对两个list排序。
- 2) 从两个list的第一个元素开始比较，若start小于end，则天上增加一架飞机，并将start进一位，反之则天上减少一架飞机，并将end进一位。记录每次增加飞机后天上飞机数量的最大值。
- 3) 当start遍历完成时，返回此时最大值。

代码如下：

```
/**
```

```
* Definition of Interval:
* public class Interval {
*     int start, end;
*     Interval(int start, int end) {
*         this.start = start;
*         this.end = end;
*     }
* }
*/

class Solution {
    /**
     * @param intervals: An interval array
     * @return: Count of airplanes are in the sky.
     */
    public int countOfAirplanes(List<Interval> airplanes) {
        // write your code here
        if(airplanes == null || airplanes.size() == 0){
            return 0;
        }

        ArrayList<Integer> start = new ArrayList<Integer>();
        ArrayList<Integer> end = new ArrayList<Integer>();
        for(int i = 0; i < airplanes.size(); i++){
            start.add(airplanes.get(i).start);
            end.add(airplanes.get(i).end);
        }

        Collections.sort(start);
        Collections.sort(end);

        int i = 0;
        int j = 0;
        int count = 0;
        int most = Integer.MIN_VALUE;
        while(i < start.size() && j < end.size()){
            if(start.get(i) < end.get(j)){
                count++;
                i++;
                most = Math.max(most, count);
            }else{
```

```
        count--;  
        j++;  
    }  
}  
return most;  
}  
}
```

# Minimum Size Subarray Sum 406

## Question

Given an array of  $n$  positive integers and a positive integer  $s$ , find the minimal length of a subarray of which the sum  $\geq s$ . If there isn't one, return -1 instead.

Example

Given the array  $[2,3,1,2,4,3]$  and  $s = 7$ , the subarray  $[4,3]$  has the minimal length under the problem constraint.

Challenge

If you have figured out the  $O(n)$  solution, try coding another solution of which the time complexity is  $O(n \log n)$ .

## Solution

遍历数组，到 $i$ 时，找到以 $i$ 元素开头的最短的子数组，若比最短子数组小则更新。

$O(n \log n)$ 的算法之后补上。

代码如下：

```
public class Solution {
    /**
     * @param nums: an array of integers
     * @param s: an integer
     * @return: an integer representing the minimum size of suba
rray
     */
    public int minimumSize(int[] nums, int s) {
        // write your code here
        if(nums == null || nums.length == 0){
            return -1;
        }

        int min = Integer.MAX_VALUE;
        for(int i = 0; i < nums.length; i++){
            int count = i;
            int sum = nums[count];
            while(sum < s){
                count++;
                if(count >= nums.length){
                    break;
                }
                sum += nums[count];
            }

            if(count >= nums.length){
                break;
            }

            min = Math.min(min, count - i + 1);
        }

        if(min == Integer.MAX_VALUE){
            return -1;
        }

        return min;
    }
}
```



# Wiggle Sort 508

## Question

Given an unsorted array `nums`, reorder it in-place such that

`nums[0] <= nums[1] >= nums[2] <= nums[3]....`

Notice

Please complete the problem in-place.

Example

Given `nums = [3, 5, 2, 1, 6, 4]`, one possible answer is `[1, 6, 2, 5, 3, 4]`.

## Solution

偶数位要小，奇数位要大。遍历数组，若走到偶数位时，若其大于前一位数，则和前一位交换；若走到奇数位，若其比前一位要小，则和前一位交换。

代码如下：



```
public class Solution {
    /**
     * @param nums a list of integer
     * @return void
     */
    public void wiggleSort(int[] nums) {
        // Write your code here
        if(nums == null || nums.length <= 1){
            return;
        }

        for(int i = 1; i < nums.length; i++){
            if((i % 2 == 1 && nums[i] < nums[i - 1]) || (i % 2 =
= 0 && nums[i] > nums[i - 1])){
                int temp = nums[i - 1];
                nums[i - 1] = nums[i];
                nums[i] = temp;
            }
        }
    }
}
```

# Reverse Pairs 532

## Question

For an array A, if  $i < j$ , and  $A[i] > A[j]$ , called  $(A[i], A[j])$  is a reverse pair. return total of reverse pairs in A.

Example

Given  $A = [2, 4, 1, 3, 5]$ ,  $(2, 1)$ ,  $(4, 1)$ ,  $(4, 3)$  are reverse pairs. return 3

## Solution

最容易想到的方法是依次查看每个元素和其后面所有元素是否是reverse pairs。但是该方法为 $O(n^2)$ ，不够快。

$O(n \log n)$ 方法：利用mergeSort的性质，即将数组分为左右两个子数组，左边数组元素index都小于右边数组元素index，同时在merge的时候会挨个比较左右两个数组元素大小，此时只要左边数组某一个元素i大于右边数组某一个元素j，则从i开始到左边数组最后一个元素和j都是reverse pairs的关系。因此，每一次merge都能寻找reverse pairs。对于每一次切割，其reverse pairs为左边数组merge找到的reverse pairs的个数 + 右边数组merge找到的reverse pairs的个数 + 左右数组merge找到的reverse pairs的个数。

代码如下：

```
public class Solution {
    /**
     * @param A an array
     * @return total of reverse pairs
     */
    public long reversePairs(int[] A) {
        // Write your code here
        if(A == null || A.length == 0){
            return 0;
        }
    }
}
```

```
        return mergeSort(A, 0, A.length - 1);
    }

    private long mergeSort(int[] A, int start, int end){
        if(start >= end){
            return 0;
        }

        long sum = 0;
        int mid = (start + end) / 2;
        sum += mergeSort(A, start, mid);
        sum += mergeSort(A, mid + 1, end);
        sum += merge(A, start, mid, end);

        return sum;
    }

    private long merge(int[] A, int start, int mid, int end){
        int[] temp = new int[A.length];
        int left = start;
        int right = mid + 1;
        int index = start;
        long sum = 0;

        while(left <= mid && right <= end){
            if(A[left] <= A[right]){
                temp[index++] = A[left++];
            }else{
                temp[index++] = A[right++];
                sum += mid - left + 1;
            }
        }

        while(left <= mid){
            temp[index++] = A[left++];
        }

        while(right <= end){
            temp[index++] = A[right++];
        }
    }
}
```

```
    }

    for(int i = start; i <= end; i++){
        A[i] = temp[i];
    }

    return sum;
}
}
```

# Load Balancer 526

## Question

Implement a load balancer for web servers. It provide the following functionality:

Add a new server to the cluster => `add(server_id)`.

Remove a bad server from the cluster => `remove(server_id)`.

Pick a server in the cluster randomly with equal probability => `pick()`.

Example

At beginning, the cluster is empty => `{}`.

```
add(1)
add(2)
add(3)
pick()
>> 1          // the return value is random, it can be either 1,
2, or 3.
pick()
>> 2
pick()
>> 1
pick()
>> 3
remove(1)
pick()
>> 2
pick()
>> 3
pick()
>> 3
```

## Solution

**add():** 用list保存加入的server，用dict记录每个server加入的位置，重复加入的server会更新为最新的位置。

**remove():** 先从dict中得到要删去server的位置，然后将list中最后一位元素移到该位置上，并将dict中该元素的位置更新，最后删去最后一位元素，并将dict中要删去的server删去。为什么要移动而不能直接删去？增加随机性？

**pick():** 用random对象随机生成，nextInt()取整之后%list.size()得到0-list.size()之间的随机数。

代码如下：

```
public class LoadBalancer {
    ArrayList<Integer> list;
    HashMap<Integer, Integer> dict;

    public LoadBalancer() {
        // Initialize your data structure here.
        list = new ArrayList<Integer>();
        dict = new HashMap<Integer, Integer>();
    }

    // @param server_id add a new server to the cluster
    // @return void
    public void add(int server_id) {
        // Write your code here
        int index = list.size();
        //用dict记录每个server加入的位置，重复加入的server会更新为最新的
        位置
        dict.put(server_id, index);
        list.add(server_id);
    }

    // @param server_id server_id remove a bad server from the c
    luster
    // @return void
    public void remove(int server_id) {
        // Write your code here
        //获得要删除元素的位置
        int index = dict.get(server_id);
```

```
//若要删除的server不存在则返回
if(index == -1){
    return;
}
int n = list.size();
//将list最后一个server移到要删除的位置上，并更新该元素在dict中记
录的位置
list.set(index, list.get(n - 1));
dict.put(list.get(n - 1), index);
//删去最后一个元素
list.remove(n - 1);
dict.remove(server_id);
}

// @return pick a server in the cluster randomly with equal
probability
public int pick() {
    // Write your code here
    int size = list.size();
    Random r = new Random();
    int index = Math.abs(r.nextInt()) % size;
    return list.get(index);
}
}
```

# Candy 412

## Question

There are N children standing in a line. Each child is assigned a rating value.

You are giving candies to these children subjected to the following requirements:

Each child must have at least one candy.

Children with a higher rating get more candies than their neighbors.

What is the minimum candies you must give?

Example

Given ratings = [1, 2], return 3.

Given ratings = [1, 1, 1], return 3.

Given ratings = [1, 2, 2], return 4. ([1,2,1]).

## Solution

我的思路：

1) 找到所有的最低点（小于等于左边元素并且小于等于右边元素），给最低点1颗糖果

2) 然后从每个最低点开始，向左右两边上升到最高点，每上升一层多给一颗糖果，两个最低点之间的最高点取给糖果多的那个值

九章答案：

1) 先给所有点1颗糖

2) 从左往右，遇到上升区间，给右边小孩比左边小孩多一颗糖

3) 从右往左，遇到上升区间，若左边小孩的糖小于等于右边小孩的糖，则给左边小孩比右边小孩多一颗糖



代码如下：

```
public class Solution {
    /**
     * @param ratings Children's ratings
     * @return the minimum candies you must give
     */
    public int candy(int[] ratings) {
        // Write your code here
        if(ratings == null || ratings.length == 0){
            return 0;
        }

        if(ratings.length == 1){
            return 1;
        }

        int[] result = new int[ratings.length];
        findValley(ratings, result);
        spreadFromValley(ratings, result);

        int sum = 0;
        for(int i = 0; i < result.length; i++){
            sum += result[i];
        }

        return sum;
    }

    private void findValley(int[] ratings, int[] result){
        for(int i = 0; i < ratings.length; i++){
            if(i == 0){
                if(ratings[i] <= ratings[i + 1]){
                    result[i] = 1;
                }
                continue;
            }

            if(i == ratings.length - 1){
                if(ratings[i] <= ratings[i - 1]){

```

```
        result[i] = 1;
    }
    continue;
}

    if(ratings[i] <= ratings[i - 1] && ratings[i] <= ratings[i + 1]){
        result[i] = 1;
    }
}

private void spreadFromValley(int[] ratings, int[] result){
    for(int i = 0; i < result.length; i++){
        if(result[i] != 1){
            continue;
        }

        int left = i - 1;
        while(left >= 0 && ratings[left] > ratings[left + 1])
        ){
            int value = result[left + 1] + 1;
            if(result[left] < value){
                result[left] = value;
            }
            left--;
        }

        int right = i + 1;
        while(right <= result.length - 1 && ratings[right] > ratings[right - 1]){
            int value = result[right - 1] + 1;
            if(result[right] < value){
                result[right] = value;
            }
            right++;
        }
    }
}
```

九章答案：

```
public class Solution {
    public int candy(int[] ratings) {
        if(ratings == null || ratings.length == 0) {
            return 0;
        }

        int[] count = new int[ratings.length];
        Arrays.fill(count, 1);
        int sum = 0;
        for(int i = 1; i < ratings.length; i++) {
            if(ratings[i] > ratings[i - 1]) {
                count[i] = count[i - 1] + 1;
            }
        }

        for(int i = ratings.length - 1; i >= 1; i--) {
            sum += count[i];
            if(ratings[i - 1] > ratings[i] && count[i - 1] <= count[i]) { // second round has two conditions
                count[i-1] = count[i] + 1;
            }
        }
        sum += count[0];
        return sum;
    }
}
```

# Merge k Sorted Arrays 486

## Question

将 k 个排序数组合并为一个大的排序数组。

### Example

给出下面的 3 个排序数组:

[1, 3, 5, 7],

[2, 4, 6],

[0, 8, 9, 10, 11]

合并后的大数组应为：

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

## Solution

与Merge K sorted Lists一样，用PriorityQueue(Heap)实现。也可以用D&C。

代码如下：

```
public class test {  
    public static void main(String[] args){  
        ArrayList<Integer> a = new ArrayList<Integer>();  
        a.add(1);  
        a.add(3);  
        a.add(5);  
        ArrayList<Integer> b = new ArrayList<Integer>();  
        b.add(2);  
        b.add(4);  
        ArrayList<ArrayList<Integer>> array = new ArrayList<ArrayList<Integer>>();  
        array.add(a);  
        array.add(b);  
    }  
}
```

```
        test t = new test();
        ArrayList<Integer> result = t.mergeKArray(array);
        for(int i : result){
            System.out.println(i);
        }
    }

    public ArrayList<Integer> mergeKArray(ArrayList<ArrayList<Integer>> arrays){
        ArrayList<Integer> merge = new ArrayList<Integer>();
        for(ArrayList<Integer> array : arrays){
            for(int i : array){
                merge.add(i);
            }
        }

        PriorityQueue<Integer> queue = new PriorityQueue<Integer>(merge);

        ArrayList<Integer> res = new ArrayList<Integer>();

        while(!queue.isEmpty()){
            res.add(queue.poll());
        }

        return res;
    }
}
```

## Related Question

之后补上

# Continuous Subarray Sum 402

## Question

Given an integer array, find a continuous subarray where the sum of numbers is the biggest. Your code should return the index of the first number and the index of the last number. (If there are duplicate answers, return anyone)

Example

Give [-3, 1, 3, -3, 4], return [1,4].

## Solution

与Maximum Subarray类似。

idea: 遍历数组，到*i*时，记录以*i*结尾的数组的最大值：

- 1) 若以*i-1*元素结尾的最大值小于0，则以*i*元素结尾的最大值为*i*元素
- 2) 若以*i-1*元素结尾的最大值大于等于0，则以*i*元素结尾的最大值为以*i-1*元素结尾的最大值加上*i*元素

代码如下：

```
// class Node{
//     int val;
//     int start;
//     int end;
//     public Node(int val, int start, int end){
//         this.val = val;
//         this.start = start;
//         this.end = end;
//     }
// }

public class Solution {
    /**
```

```

    * @param A an integer array
    * @return A list of integers includes the index of the first number and the index of the last number
    */
    public ArrayList<Integer> continuousSubarraySum(int[] A) {
        // Write your code here
        ArrayList<Integer> result = new ArrayList<Integer>();
        result.add(-1);
        result.add(-1);
        if(A == null || A.length == 0){
            return result;
        }

        // Node[] DP = new Node[A.length];
        // DP[0] = new Node(A[0], 0, 0);
        // for(int i = 1; i < A.length; i++){
        //     if(DP[i - 1].val + A[i] > A[i]){
        //         DP[i] = new Node(DP[i - 1].val + A[i], DP[i - 1].start, i);
        //     }else{
        //         DP[i] = new Node(A[i], i, i);
        //     }
        // }

        // Node max = DP[0];
        // for(int i = 1; i < DP.length; i++){
        //     if(DP[i].val > max.val){
        //         max = DP[i];
        //     }
        // }

        int start = 0;
        int end = 0;
        int sum = 0;
        int max = Integer.MIN_VALUE;
        for(int i = 0; i < A.length; i++){
            if(sum < 0){
                sum = A[i];
                start = end = i;
            }else{

```

```
        sum = sum + A[i];
        end = i;
    }
    if(sum >= max){
        max = sum;
        result.set(0, start);
        result.set(1, end);
    }
}

// result.add(max.start);
// result.add(max.end);
return result;
}
}
```



# Continuous Subarray Sum II 403

## Question

Given an circular integer array (the next element of the last element is the first element), find a continuous subarray in it, where the sum of numbers is the biggest. Your code should return the index of the first number and the index of the last number.

If duplicate answers exist, return any of them.

Example

Give [3, 1, -100, -3, 4], return [4,1].

## Solution

这道题和continuous subarray sum I思路一样，不同之处在于可以从成环，因此解法稍有不同。总共分为两种情况讨论：

1. 首先遍历数组找出没有环的情况下最大的子数组之和，方法和I相同。
2. 然后找有环情况下最大子数组之和。做法为，寻找数组中最小和的连续子数组，然后用整个数组之和减去这个最小和的连续子数组剩下的就是该情况下的最大和的子数组。将该情况下得到的最大和与1中得到的最大和相比，取和更大的那种即可。
3. 几个tips：1) 在1中遍历时，可以顺便求出整个数组的和。2) 2中要考虑一种情况为整个数组全部为负数，这样需要减去的是整个数组，因此返回1中的结果。3) 根据2中求的最小数组，最大数组的index因该为 $end + 1$ 到 $start - 1$ ，但是为了防止出现最小数组包含头尾的情况（即 $end + 1$ 或 $start - 1$ 越界），index的表示方法为 $(end + 1) \% A.length$ 和 $(start - 1 + A.length) \% A.length$ 。

代码如下：

```
public class Solution {  
    /**
```

```
* @param A an integer array
* @return A list of integers includes the index of the first number and the index of the last number
*/
public ArrayList<Integer> continuousSubarraySumII(int[] A) {
    // Write your code here
    if(A == null || A.length == 0){
        return new ArrayList<Integer>();
    }

    ArrayList<Integer> res = new ArrayList<Integer>();
    res.add(0);
    res.add(0);
    int total = 0;
    int start = 0;
    int end = 0;
    int local = 0;
    int global = Integer.MIN_VALUE;
    //先找不循环情况下最大子数组
    for(int i = 0; i < A.length; i++){
        total += A[i];
        if(local > 0){
            local += A[i];
            end = i;
        }else{
            local = A[i];
            start = end = i;
        }
        if(local > global){
            global = local;
            res.set(0, start);
            res.set(1, end);
        }
    }

    start = 0;
    end = -1;
    local = 0;
    //找最小子数组，数组总和减去最小子数组即为又循环情况下最大子数组
    for(int i = 0; i < A.length; i++){
```

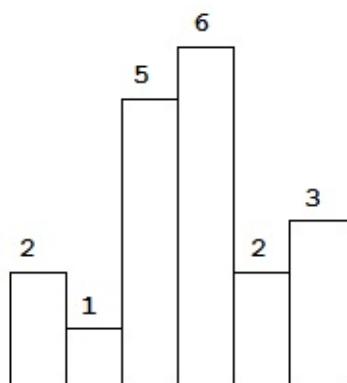
```
        if(local <= 0){
            local += A[i];
            end = i;
        }else{
            local = A[i];
            start = end = i;
        }
        //若最小数组为整个数组（即所有元素为负），则返回第一种情况结果
        if(start == 0 && end == A.length - 1){
            continue;
        }
        //比较又循环情况和无循环情况大小，取大者
        if(total - local > global){
            global = total - local;
            //为了防止出现最小子数组包含头尾而导致越界
            res.set(0, (end + 1) % A.length);
            res.set(1, (start - 1 + A.length) % A.length);
        }
    }

    return res;
}
}
```

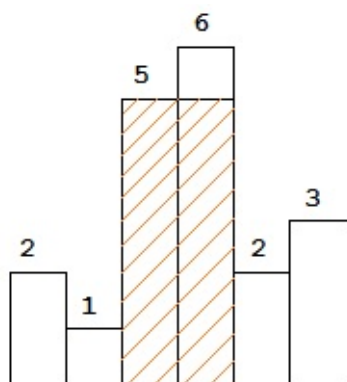
# Largest Rectangle in Histogram 122

## Question

Given  $n$  non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.



Above is a histogram where width of each bar is 1, given height = [2,1,5,6,2,3].



The largest rectangle is shown in the shaded area, which has area = 10 unit.

Example

Given height = [2,1,5,6,2,3],

return 10.

## Solution

用`stack`存储高度的`index`，当当前高度大于栈顶高度时，直接将当前高度入栈；当前高度小于等于栈顶高度时，将栈顶元素出栈，再次判断直到栈中没有比当前高度大的元素，再将当前高度入栈。这样，栈中每个元素出栈时，其左边的元素为该元素在数组中左边第一个小于该元素的元素，而导致该元素出栈的元素（即此轮的当前高度）为该元素在数组中右边第一个小于该元素的元素，因此用右边界减去左边界再减1为满足该元素高度的最大宽度，再乘以该高度为该元素高度的最大面积。特殊情况为该出栈元素左边没有元素，表示从数组的第一个元素到该元素之间的元素都比该元素高度大，所以该元素的最大宽度为从数组开头到其右边界。当数组所有元素都遍历过之后，若`stack`中还有元素剩下，则将当前元素高度赋值为-1，这样`stack`中所有元素都会出栈。

代码如下：

```
public class Solution {
    /**
     * @param height: A list of integer
     * @return: The area of largest rectangle in the histogram
     */
    public int largestRectangleArea(int[] height) {
        // write your code here

        if(height == null || height.length == 0){
            return 0;
        }

        Stack<Integer> stack = new Stack<Integer>();

        int max = -1;
        for(int i = 0; i <= height.length; i++){
            int current = (i == height.length) ? -1 : height[i];
            while(!stack.isEmpty() && current <= height[stack.peak()]){
                int h = height[stack.pop()];
                int w = (stack.isEmpty()) ? i : i - stack.peek() - 1;
                max = Math.max(max, w * h);
            }
            stack.push(i);
        }

        return max;
    }
}
```

# Rotate Array (LeetCode) 189

## Question

Rotate an array of  $n$  elements to the right by  $k$  steps.

For example, with  $n = 7$  and  $k = 3$ , the array  $[1,2,3,4,5,6,7]$  is rotated to  $[5,6,7,1,2,3,4]$ .

## Solution

解法一：普通解法，依次向右平移 $k$ 步。先将最后 $k$ 个元素记录下来，再将前 $n-k$ 个元素向右平移 $k$ 步，最后将之前记录下来的 $k$ 个元素填入原来数组的前 $k$ 个位置。

解法二：先将前 $n-k$ 个元素反转，再将后 $k$ 个元素反转，最后将整个数组反转。这里有个小技巧，交换两个元素 $a,b$ 时，可以用

```
a ^= b;  
b ^= a;  
a ^= b;
```

结果和传统的用`temp`交换的方法是一样的。

代码如下：

```
public class Solution {  
    public void rotate(int[] nums, int k) {  
        if (nums == null || nums.length == 0 || k == 0) {  
            return;  
        }  
  
        //reverse version  
        // int n = nums.length;  
        // k %= n;  
  
        // reverse(nums, 0, n - k - 1);  
    }  
}
```

```
// reverse(nums, n - k, n - 1);
// reverse(nums, 0, n - 1);

//normal version
int step = k % nums.length;
int[] tmp = new int[step];
for(int i = 0; i < step; i++){
    tmp[i] = nums[nums.length - step + i];
}
for(int i = nums.length - step - 1; i >= 0; i--){
    nums[i + step] = nums[i];
}
for(int i = 0; i < step; i++){
    nums[i] = tmp[i];
}
}

private void reverse (int[] nums, int start, int end) {
    if (start >= end) {
        return;
    }

    while (start < end) {
        // int temp = nums[start];
        // nums[start] = nums[end];
        // nums[end] = temp;
        nums[start] ^= nums[end];
        nums[end] ^= nums[start];
        nums[start] ^= nums[end];
        start++;
        end--;
    }
}
}
```



# Linked List

# Swap Two Nodes in Linked List 511

## Question

Given a linked list and two values  $v_1$  and  $v_2$ . Swap the two nodes in the linked list with values  $v_1$  and  $v_2$ . It's guaranteed there is no duplicate values in the linked list. If  $v_1$  or  $v_2$  does not exist in the given linked list, do nothing.

Notice

You should swap the two nodes with values  $v_1$  and  $v_2$ . Do not directly swap the values of the two nodes.

Example

Given 1->2->3->4->null and  $v_1 = 2$ ,  $v_2 = 4$ .

Return 1->4->3->2->null.

## Solution

先找第一个node和它parent，再找第二个node和它parent，再进行交换。用一个dummy node记录head，在寻找过程中一旦发现无法找到其中一个node，就返回head。最后在交换时，需要判断第一个node的下一个节点是不是第二个node。

代码如下：

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    /**
```

```
* @param head a ListNode
* @param v1 an integer
* @param v2 an integer
* @return a new head of singly-linked list
*/
public ListNode swapNodes(ListNode head, int v1, int v2) {
    // Write your code here
    if(head == null){
        return head;
    }

    ListNode dummy = new ListNode(0);
    dummy.next = head;
    head = dummy;
    //先找第一个
    while(head.next != null && head.next.val != v1 && head.next.val != v2){
        head = head.next;
    }

    if(head.next == null){
        return dummy.next;
    }

    ListNode firstParent = head;
    ListNode first = head.next;
    //再找第二个
    if(head.next.val == v1){
        while(head.next != null && head.next.val != v2){
            head = head.next;
        }
    }else{
        while(head.next != null && head.next.val != v1){
            head = head.next;
        }
    }

    if(head.next == null){
        return dummy.next;
    }
}
```

```
    ListNode secondParent = head;
    ListNode second = head.next;
    ListNode secondNext = second.next;

    //判断first和second是不是相连
    if(first.next == second){
        firstParent.next = second;
        second.next = first;
        first.next = secondNext;
    }else{
        firstParent.next = second;
        secondParent.next = first;
        second.next = first.next;
        first.next = secondNext;
    }

    return dummy.next;
}
}
```

# Intersection of Two Linked Lists 380

## Question

Write a program to find the node at which the intersection of two singly linked lists begins.

Notice

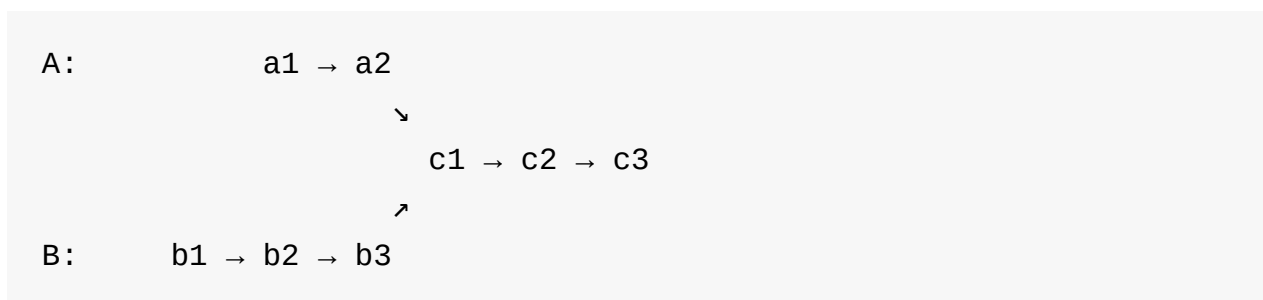
If the two linked lists have no intersection at all, return null.

The linked lists must retain their original structure after the function returns.

You may assume there are no cycles anywhere in the entire linked structure.

Example

The following two linked lists:



begin to intersect at node c1.

Challenge

Your code should preferably run in  $O(n)$  time and use only  $O(1)$  memory.

## Solution

可以将问题转换为寻找有环的Linked List的环的起点。

1) 找到第一个Linked List的最后一个node，将其和第二个Linked List的head相连

2) 若两个Linked List相交，则一定会形成一个环（若无环则证明不相交，返回null）

3) 再以第一个Linked List的head为起点寻找环的起点即可

4) 为保持两个Linked List结构，最后要将第一个Linked List的最后一个节点指向null

代码如下：

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param headA: the first list
     * @param headB: the second list
     * @return: a ListNode
     */
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        // Write your code here
        if(headA == null || headB == null){
            return null;
        }

        ListNode curt = headA;
        while(curt.next != null){
            curt = curt.next;
        }

        curt.next = headB;

        ListNode intersection = listCycle(headA);
    }
}
```

```
        curt.next = null;

        return intersection;
    }

    private ListNode listCycle(ListNode head){
        ListNode slow = head;
        ListNode fast = head.next;

        while(fast != slow){
            if(fast == null || fast.next == null){
                return null;
            }
            slow = slow.next;
            fast = fast.next.next;
        }

        slow = head;
        fast = fast.next;
        while(slow != fast){
            slow = slow.next;
            fast = fast.next;
        }

        return slow;
    }
}
```

## Related Question

[Linked List Cycle II 103](#)

# Reverse Nodes in k-Group 450

## Question

Given a linked list, reverse the nodes of a linked list  $k$  at a time and return its modified list.

If the number of nodes is not a multiple of  $k$  then left-out nodes in the end should remain as it is.

You may not alter the values in the nodes, only nodes itself may be changed. Only constant memory is allowed.

### Example

Given this linked list: 1->2->3->4->5

For  $k = 2$ , you should return: 2->1->4->3->5

For  $k = 3$ , you should return: 3->2->1->4->5

## Solution

思想很简单，就是从第 $i$ 个元素开始，每次往后走 $k$ 步，找到第 $i+k$ 个元素，然后把第 $i+1$ 个元素到第 $i+k$ 个元素全部反转相连，最后再将第 $i$ 个元素和第 $i+k$ 个元素相连，以及第 $i+1$ 个元素和第 $i+k+1$ 个元素相连。再将刚才的第 $i+1$ 个元素作为新的第“ $i$ ”个元素重复之前步骤。如果某一次移动不足 $k$ 次就到尾部，则停止。用符号表示就是

Change head->n1->...->nk->next.. to head->nk->...->n1->next..。

反转一个linked list的套路为用三个指针，一个记录pre，一个记录curt，一个记录next，将curt指向pre，然后再将pre，curt，next分别向右移动一步（pre = curt, curt = next, next = next.next）。

代码如下：

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
```



```
*     int val;
*     ListNode next;
*     ListNode(int x) { val = x; }
* }
*/
public class Solution {
    /**
     * @param head a ListNode
     * @param k an integer
     * @return a ListNode
     */
    public ListNode reverseKGroup(ListNode head, int k) {
        if (head == null || k <= 1) {
            return head;
        }

        ListNode dummy = new ListNode(0);
        dummy.next = head;

        head = dummy;
        while (head.next != null) {
            head = reverseNextK(head, k);
        }

        return dummy.next;
    }

    // reverse head->n1->...->nk->next..
    // to head->nk->...->n1->next..
    // return n1
    private ListNode reverseNextK(ListNode head, int k) {
        // check there is enough nodes to reverse
        ListNode next = head; // next is not null
        for (int i = 0; i < k; i++) {
            if (next.next == null) {
                return next;
            }
            next = next.next;
        }
    }
}
```

```
        // reverse
        ListNode n1 = head.next;
        ListNode prev = head, curt = n1;
        for (int i = 0; i < k; i++) {
            ListNode temp = curt.next;
            curt.next = prev;
            prev = curt;
            curt = temp;
        }

        n1.next = curt;
        head.next = prev;
        return n1;
    }
}
```

# Stack



```

ger,
    *      // rather than a nested list.
    *      public boolean isInteger();
    *
    *      // @return the single integer that this NestedInteger holds,
ds,
    *      // if it holds a single integer
    *      // Return null if this NestedInteger holds a nested list
    *      public Integer getInteger();
    *
    *      // @return the nested list that this NestedInteger holds,
    *      // if it holds a nested list
    *      // Return null if this NestedInteger holds a single integer
er
    *      public List<NestedInteger> getList();
    * }
    */
import java.util.Iterator;

public class NestedIterator implements Iterator<Integer> {
    List<NestedInteger> list;
    Stack<NestedInteger> stack = new Stack<NestedInteger>();

    public NestedIterator(List<NestedInteger> nestedList) {
        // Initialize your data structure here.
        list = nestedList;
        for(int i = list.size() - 1; i >= 0; i--){
            stack.push(list.get(i));
        }
    }

    // @return {int} the next element in the iteration
    @Override
    public Integer next() {
        // Write your code here
        return stack.pop().getInteger();
    }

    // @return {boolean} true if the iteration has more element
    or false

```

```
@Override
public boolean hasNext() {
    // Write your code here
    while(!stack.isEmpty()){
        if(stack.peek().isInteger()){
            return true;
        }
        NestedInteger curt = stack.pop();
        for(int i = curt.getList().size() - 1; i >= 0; i--){
            stack.push(curt.getList().get(i));
        }
    }
    return false;
}

@Override
public void remove() {
}
}

/**
 * Your NestedIterator object will be instantiated and called as
 * such:
 * NestedIterator i = new NestedIterator(nestedList);
 * while (i.hasNext()) v.add(i.next());
 */
```

# Valid Parentheses 423

## Question

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

Example

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(" and "([)]" are not.

## Solution

用stack实现。遍历string中所有字符，遇到前括号就压入栈中，遇到后括号就看栈顶元素（与该后括号距离最近的前括号）是否和该后括号配对，配对则返回true，不配对或此时stack为空则返回false。

代码如下：

```
public class Solution {
    public boolean isValidParentheses(String s) {
        Stack<Character> stack = new Stack<Character>();
        for (Character c : s.toCharArray()) {
            if ("({[".contains(String.valueOf(c))) {
                stack.push(c);
            } else {
                if (!stack.isEmpty() && is_valid(stack.peek(), c))
                ) {
                    stack.pop();
                } else {
                    return false;
                }
            }
        }
        return stack.isEmpty();
    }

    private boolean is_valid(char c1, char c2) {
        return (c1 == '(' && c2 == ')') || (c1 == '{' && c2 == '
    }')
        || (c1 == '[' && c2 == ']');
    }
}
```



# Evaluate Reverse Polish Notation 424

## Question

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are +, -, \*, /. Each operand may be an integer or another expression.

Example

`["2", "1", "+", "3", "*"] -> ((2 + 1) * 3) -> 9`

`["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6`

## Solution

完全按照RPN的定义即可。具体过程wikipedia上说的非常清楚。

例子：

`5 1 2 + 4 × + 3 -`

Input	Action	Stack	Notes
5	Operand	5	Push onto stack.
1	Operand	1 5	Push onto stack.
2	Operand	2 1 5	Push onto stack.
+	Operator	3 5	Pop the two operands (1, 2), calculate (1 + 2 = 3) and push onto stack.
4	Operand	4 3 5	Push onto stack.
×	Operator	12 5	Pop the two operands (3, 4), calculate (3 * 4 = 12) and push onto stack.
+	Operator	17	Pop the two operands (5, 12), calculate (5 + 12 = 17) and push onto stack.
3	Operand	3 17	Push onto stack.
-	Operator	14	Pop the two operands (17, 3), calculate (17 - 3 = 14) and push onto stack.
	Result	14	

代码如下：

```
public class Solution {
    /**
     * @param tokens The Reverse Polish Notation
     * @return the value
     */
    public int evalRPN(String[] tokens) {
        // Write your code here
        if(tokens == null || tokens.length == 0){
            return 0;
        }

        Stack<Integer> stack = new Stack<Integer>();

        for(int i = 0; i < tokens.length; i++){
            if("+-*/*".contains(tokens[i])){
                // if(stack.size() < 2){
                //     return -1;
                // }
                int a = stack.pop();
                int b = stack.pop();
                if(tokens[i].equals("+")){
                    stack.push(b + a);
                }
                if(tokens[i].equals("-")){
                    stack.push(b - a);
                }
                if(tokens[i].equals("*")){
                    stack.push(b * a);
                }
                if(tokens[i].equals("/")){
                    stack.push(b / a);
                }
            }else{
                stack.push(Integer.valueOf(tokens[i]));
            }
        }

        return stack.pop();
    }
}
```

```
}
```

# Simplify Path 421

## Question

Given an absolute path for a file (Unix-style), simplify it.

Example

`"/home/", => "/home"`

`"/a/./b/../../c/", => "/c"`

## Solution

Unix路径简化的依据是：

当遇到`"/.."`则需要返回上级目录，需检查上级目录是否为空。

当遇到`"/."`则表示是本级目录，无需做任何特殊操作。

当遇到`"/"`则表示是本级目录，无需做任何操作。

当遇到其他字符则表示是文件夹名，无需简化。

当字符串是空或者遇到`"/.."`，则需要返回一个`"/"`。

当遇见`"/a//b"`，则需要简化为`"/a/b"`。

因此，可以用两个stack来实现。

第一个栈用于保存所有合法的文件夹名。首先将path根据`"/"`进行划分，对于所有划分出来的字符串，

1. 若等于`""`（例如`"/."`划分后就是`""`和`"/."`）或者`"/."`则什么都不做，
2. 若等于`"/.."`则将栈顶元素出栈（相当于返回上级目录，栈为空则什么都不做），
3. 否则入栈（合法文件夹名）。

遍历完所有划分出的字符串后，若栈为空，则返回"/"，否则要按照顺序返回栈中所有文件夹目录名。但是因为栈，所以栈底元素应该在最前名，所以用第二个栈来将第一个栈中元素取相反顺序后输出。也可以不用第二个stack，直接用string，每次加到最前面，最后将最后一个字符（"/"）删去即可。

代码如下：

```
public class Solution {
    /**
     * @param path the original path
     * @return the simplified path
     */
    public String simplifyPath(String path) {
        // Write your code here
        if(path == null || path.length() == 0){
            return null;
        }

        String[] array = path.split("/");

        Stack<String> stack = new Stack<String>();
        for(int i = 0; i < array.length; i++){
            if(array[i].equals(".") || array[i].equals("")){
                continue;
            }

            if(array[i].equals("..")){
                if(!stack.isEmpty()){
                    stack.pop();
                }
                continue;
            }

            stack.push(array[i]);
        }

        if(stack.isEmpty()){
            return "/";
        }
    }
}
```

```
Stack<String> temp = new Stack<String>();
while(!stack.isEmpty()){
    temp.push(stack.pop());
}

String res = "";
while(!temp.isEmpty()){
    res = res + "/" + temp.pop();
}

return res;
}
}
```

# Max Tree 126

## Question

Given an integer array with no duplicates. A max tree building on this array is defined as follow:

The root is the maximum number in the array

The left subtree and right subtree are the max trees of the subarray divided by the root number.

Construct the max tree by the given array.

Example

Given [2, 5, 6, 0, 3, 1], the max tree constructed by this array is:

```
6
```

```
/\ 5 3 // \ 2 0 1
```

Challenge

$O(n)$  time and memory.

## Solution

本题其实是构建笛卡树（Cartesian tree，

[https://en.wikipedia.org/wiki/Cartesian\\_tree](https://en.wikipedia.org/wiki/Cartesian_tree)），经典方法是用单调栈（单调递减栈）。我们堆栈里存放的树，只有左子树，没有右子树，且根节点最大。时间复杂度为 $O(n)$ 。

1. 如果新来一个数，比堆栈顶的树根的数小，则把这个数作为一个单独的节点压入堆栈。

2. 否则，不断从堆栈里弹出树，新弹出的树以旧弹出的树为右子树，连接起来，直到目前堆栈顶的树根的数大于新来的数。然后，弹出的那些数，已经形成了一个树，这个树作为新节点的左子树，把这个新树压入堆栈。

这样的堆栈是单调的，越靠近堆栈顶的数越小。最后还要按照（2）的方法，把所有树弹出来，每个旧树作为新树的右子树。

如果用递归的方法建树，时间复杂度会退化到 $O(n^2)$ 。

代码如下：

Stack  $O(n)$ ：



```

public class Solution {
    /**
     * @param A: Given an integer array with no duplicates.
     * @return: The root of max tree.
     */
    public TreeNode maxTree(int[] A) {
        // write your code here
        if(A == null || A.length == 0){
            return null;
        }

        Stack<TreeNode> stack = new Stack<TreeNode>();
        for(int i = 0; i <= A.length; i++){
            TreeNode newNode = i == A.length? new TreeNode(Integer.MAX_VALUE) : new TreeNode(A[i]);
            if(!stack.isEmpty() && stack.peek().val < newNode.val){
                TreeNode curt = stack.pop();
                TreeNode root = curt;
                while(!stack.isEmpty() && stack.peek().val < newNode.val){
                    root = stack.pop();
                    root.right = curt;
                    curt = root;
                }
                newNode.left = root;
            }
            stack.push(newNode);
        }

        return stack.peek().left;
    }
}

```

Recursive  $O(n^2)$ :

```

public class Solution {
    /**
     * @param A: Given an integer array with no duplicates.

```

```
* @return: The root of max tree.
*/
// class Node{
//     int val;
//     int index;
//     public Node(int val, int index){
//         this.val = val;
//         this.index = index;
//     }
// }
// public TreeNode maxTree(int[] A) {
//     // write your code here
//     if(A == null || A.length == 0){
//         return null;
//     }

//     return helper(A, 0, A.length - 1);
// }

// private TreeNode helper(int[] A, int start, int end){
//     if(start > end){
//         return null;
//     }

//     if(start == end){
//         return new TreeNode(A[start]);
//     }

//     Stack<Node> stack = new Stack<Node>();
//     for(int i = start; i <= end; i++){
//         if(!stack.isEmpty() && stack.peek().val > A[i]){
//             continue;
//         }
//         stack.push(new Node(A[i], i));
//     }

//     TreeNode root = new TreeNode(stack.peek().val);

//     root.left = helper(A, start, stack.peek().index - 1);
//     root.right = helper(A, stack.peek().index + 1, end);
// }
```

```
        //      return root;  
        // }  
    }  
}
```

# Expression Tree Build 367

## Question

The structure of Expression Tree is a binary tree to evaluate certain expressions. All leaves of the Expression Tree have an number string value. All non-leaves of the Expression Tree have an operator string value.

Now, given an expression array, build the expression tree of this expression, return the root of this expression tree.

Clarification

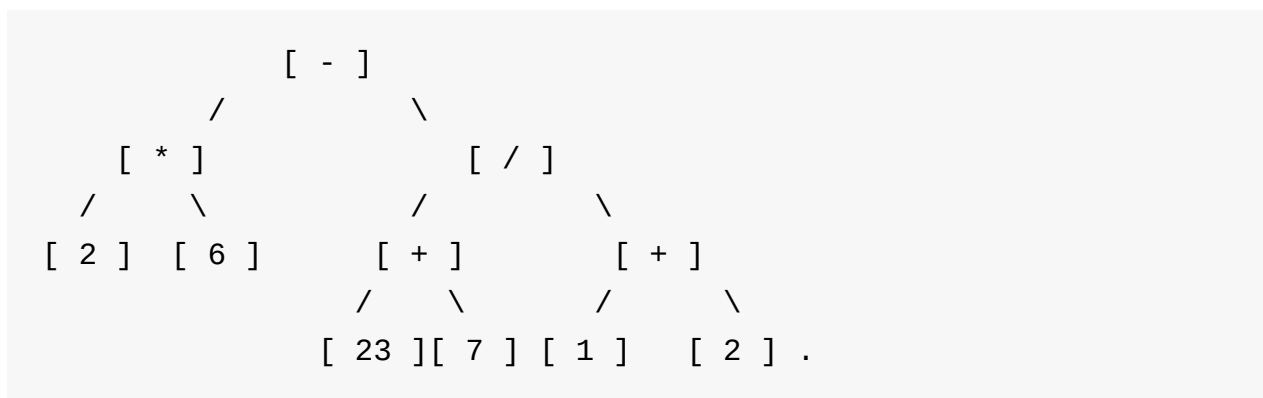
See wiki:

Expression Tree

Example

For the expression  $(26-(23+7)/(1+2))$  (which can be represented by `["2" "" "6" "-" "(" "23" "+" "7" ")" "/" "(" "1" "+" "2" ")""]`).

The expression tree will be like



After building the tree, you just need to return root node [-].

## Solution

观察example，可以看出所有叶节点都为数字。如果给每个元素赋予一个优先级，和  $\wedge$  为2，+ 和 - 为1，数字为极大值，然后规定优先级越大的越在下，越小的越在上。这样，这道题就转化为构建\*Min Tree，和之前的Max Tree做法类似，只是这里维持的是一个递增栈。同时，当遇见“(”时，提高优先级，遇见“)”时，降低优先级。

1. 遍历数组，给每个新来的元素赋予一个val值用以比较优先级。 $\wedge$  和  $\wedge$  为2，+ 和 - 为1，数字为极大值。
2. 此时看栈顶元素（若栈为空则直接加入）。为了维持一个递增栈，若栈顶元素比新来元素val大（或相等），则出栈；若栈顶元素比新来元素val小，则break。
3. 若2中栈顶元素出栈，此时若栈为空，则将出栈元素作为新来元素的左节点，并将新来元素加入栈中；若不为空，看新栈顶元素，若新栈顶元素比新来元素val小，则将出栈元素作为新来元素的左孩子，并将新来元素加入栈中；若新栈顶元素比新来元素val大（或相等），则将出栈元素作为新栈顶元素的右节点，重复2-3，直到栈为空或者栈顶元素比新来元素要小，将新来元素加入栈中。
4. tips：在遍历完整个数组后，多加一个值，将其val赋值为极小，这样所有元素都会出栈并构建成完整的树。

代码如下：

```
/**
 * Definition of ExpressionTreeNode:
 * public class ExpressionTreeNode {
 *     public String symbol;
 *     public ExpressionTreeNode left, right;
 *     public ExpressionTreeNode(String symbol) {
 *         this.symbol = symbol;
 *         this.left = this.right = null;
 *     }
 * }
 */

public class Solution {
    /**
     * @param expression: A string array
     * @return: The root of expression tree
     */
}
```

```
*/
class TreeNode{
    int val;
    ExpressionTreeNode root;
    public TreeNode(int val, String s){
        this.val = val;
        root = new ExpressionTreeNode(s);
    }
}

public int getPriority(String s, int base){
    if(s.equals("-") || s.equals("+")){
        return 1 + base;
    }
    if(s.equals("*") || s.equals("/")){
        return 2 + base;
    }
    return Integer.MAX_VALUE;
}

public ExpressionTreeNode build(String[] expression) {
    // write your code here
    if(expression == null || expression.length == 0){
        return null;
    }

    int val = 0;
    int base = 0;
    Stack<TreeNode> stack = new Stack<TreeNode>();
    for(int i = 0; i <= expression.length; i++){
        if(i != expression.length){
            if(expression[i].equals("(")){
                base += 10;
                continue;
            }
            if(expression[i].equals(")")){
                base -= 10;
                continue;
            }
            val = getPriority(expression[i], base);
        }
    }
}
```

```
    }

    TreeNode right = i == expression.length? new TreeNod
e(Integer.MIN_VALUE, "") : new TreeNode(val, expression[i]);
    while(!stack.isEmpty()){
        if(stack.peek().val >= right.val){
            TreeNode nodeNow = stack.pop();
            if(stack.isEmpty()){
                right.root.left = nodeNow.root;
            }else{
                TreeNode left = stack.peek();
                if(left.val < right.val){
                    right.root.left = nodeNow.root;
                }else{
                    left.root.right = nodeNow.root;
                }
            }
        }else{
            break;
        }
    }
    stack.push(right);
}

return stack.peek().root.left;
}
}
```

## Related Question

[Max Tree 126](#)

# Mock Hanoi Tower by Stacks 227

## Question

In the classic problem of Towers of Hanoi, you have 3 towers and N disks of different sizes which can slide onto any tower. The puzzle starts with disks sorted in ascending order of size from top to bottom (i.e., each disk sits on top of an even larger one). You have the following constraints:

Only one disk can be moved at a time.

A disk is slid off the top of one tower onto the next tower.

A disk can only be placed on the top of a larger disk.

Write a program to move the disks from the first tower to the last using stacks.

## Solution

**add** : 如果栈顶元素大于要加入元素，则加入

**moveTopTo** : 如果目标柱子的栈顶元素大于要移动元素，则加入目标柱子栈

**moveDisks** : 如果要移动盘子数量 $n$ 大于0，则递归地将 $n-1$ 个盘子通过目标柱子移到buffer柱子，然后将最后一个盘子移到目标柱子，最后再递归地将buffer柱子上的 $n-1$ 个盘子通过原来的柱子移到目标柱子

代码如下：

```
public class Tower {
    private Stack<Integer> disks;
    // create three towers (i from 0 to 2)
    public Tower(int i) {
        disks = new Stack<Integer>();
    }

    // Add a disk into this tower
    public void add(int d) {
```



```
        if (!disks.isEmpty() && disks.peek() <= d) {
            System.out.println("Error placing disk " + d);
        } else {
            disks.push(d);
        }
    }

    // @param t a tower
    // Move the top disk of this tower to the top of t.
    public void moveTopTo(Tower t) {
        // Write your code here
        if(t.disks.size() != 0 && t.disks.peek() < this.disks.peak()){
            System.out.println("Error Moving");
            return;
        }
        t.disks.push(this.disks.pop());
    }

    // @param n an integer
    // @param destination a tower
    // @param buffer a tower
    // Move n Disks from this tower to destination by buffer tower
    public void moveDisks(int n, Tower destination, Tower buffer) {
        // Write your code here
        if(n == 0){
            return;
        }
        moveDisks(n - 1, buffer, destination);
        moveTopTo(destination);
        buffer.moveDisks(n - 1, destination, this);
    }

    public Stack<Integer> getDisks() {
        return disks;
    }
}
/**
```

```
* Your Tower object will be instantiated and called as such:  
* Tower[] towers = new Tower[3];  
* for (int i = 0; i < 3; i++) towers[i] = new Tower(i);  
* for (int i = n - 1; i >= 0; i--) towers[0].add(i);  
* towers[0].moveDisks(n, towers[2], towers[1]);  
* print towers[0], towers[1], towers[2]  
*/
```

# Hash Table

# Happy Number 488

## Question

Write an algorithm to determine if a number is happy.

A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers.

Example

19 is a happy number

$$1^2 + 9^2 = 82$$

$$8^2 + 2^2 = 68$$

$$6^2 + 8^2 = 100$$

$$1^2 + 0^2 + 0^2 = 1$$

## Solution

将每一位的平方相加得到一个新的数n，用一个HashSet记录所有曾经得到过的数。若有重复，则返回false。

代码如下：

```
public class Solution {  
    /**  
     * @param n an integer  
     * @return true if this is a happy number or false  
     */  
    public boolean isHappy(int n) {  
        // Write your code here  
        if(n == 0){  
            return false;  
        }  
  
        HashSet<Integer> nums = new HashSet<Integer>();  
  
        while(n != 1){  
            if(nums.contains(n)){  
                return false;  
            }  
            nums.add(n);  
            int sum = 0;  
            while(n != 0){  
                sum += (n % 10) * (n % 10);  
                n = n / 10;  
            }  
            n = sum;  
        }  
  
        return true;  
    }  
}
```

# Intersection of Two Arrays 547

## Question

Given two arrays, write a function to compute their intersection.

Notice

Each element in the result must be unique.

The result can be in any order.

## Solution

找两个数组中重复的元素，且在答案数组中重复元素只出现一次。

第一种想法是先对两个数组排序，然后从头开始依次比较两个数组的值，直到一个数组用尽。

1. 若相等，并且答案数组中没有，则加入答案数组
2. 若不想等，则将小的增大

第二种想法是利用HashSet。先将一个数组加入到一个第一个HashSet中，然后遍历另外一个数组，若其中元素在第一个HashSet中，则将相等的元素加入到另一个作为答案的HashSet中。

第三中方法可以先将一个数组排序后，遍历第二个数组，用binary search的方法在第一个数组中寻找第二个数组的元素。

第一个方法 $O(n\log n)$ ，但是若数组已经排好序，则 $O(n)$ 。

第二个方法时间 $O(n)$ ，空间 $O(n)$ 。

第三个方法时间 $O(mn\log n)$ 。

代码如下：

Sort

```
public class Solution {
    /**
     * @param nums1 an integer array
     * @param nums2 an integer array
     * @return an integer array
     */
    public int[] intersection(int[] nums1, int[] nums2) {
        // Write your code here
        Arrays.sort(nums1);
        Arrays.sort(nums2);

        ArrayList<Integer> temp = new ArrayList<Integer>();
        int i = 0; int j = 0;
        while(i < nums1.length && j < nums2.length){
            if(nums1[i] == nums2[j]){
                if(temp.size() == 0 || temp.get(temp.size() - 1)
!= nums1[i]){
                    temp.add(nums1[i]);
                }
                i++;
                j++;
            }else if(nums1[i] < nums2[j]){
                i++;
            }else{
                j++;
            }
        }

        int[] res = new int[temp.size()];
        int count = 0;
        for(int item : temp){
            res[count++] = item;
        }

        return res;
    }
}
```

```
public class Solution {
    /**
     * @param nums1 an integer array
     * @param nums2 an integer array
     * @return an integer array
     */
    public int[] intersection(int[] nums1, int[] nums2) {
        // Write your code here
        if(nums1 == null || nums1.length == 0 || nums2 == null ||
        | nums2.length == 0){
            return new int[]{};
        }

        HashSet<Integer> num = new HashSet<Integer>();
        for(int i = 0; i < nums1.length; i++){
            num.add(nums1[i]);
        }

        HashSet<Integer> res = new HashSet<Integer>();
        for(int i = 0; i < nums2.length; i++){
            if(num.contains(nums2[i])){
                res.add(nums2[i]);
            }
        }

        int[] result = new int[res.size()];
        int count = 0;
        for(int i : res){
            result[count++] = i;
        }

        return result;
    }
}
```

## Sort + Binary Search

```
public class Solution {
    /**
```



```
* @param nums1 an integer array
* @param nums2 an integer array
* @return an integer array
*/
public int[] intersection(int[] nums1, int[] nums2) {
    if (nums1 == null || nums2 == null) {
        return null;
    }

    HashSet<Integer> set = new HashSet<>();

    Arrays.sort(nums1);
    for (int i = 0; i < nums2.length; i++) {
        if (set.contains(nums2[i])) {
            continue;
        }
        if (binarySearch(nums1, nums2[i])) {
            set.add(nums2[i]);
        }
    }

    int[] result = new int[set.size()];
    int index = 0;
    for (Integer num : set) {
        result[index++] = num;
    }

    return result;
}

private boolean binarySearch(int[] nums, int target) {
    if (nums == null || nums.length == 0) {
        return false;
    }

    int start = 0, end = nums.length - 1;
    while (start + 1 < end) {
        int mid = (end - start) / 2 + start;
        if (nums[mid] == target) {
            return true;
        }
    }
}
```

```
        }
        if (nums[mid] < target) {
            start = mid;
        } else {
            end = mid;
        }
    }

    if (nums[start] == target) {
        return true;
    }
    if (nums[end] == target) {
        return true;
    }

    return false;
}
}
```

# Intersection of Two Arrays II 548

## Question

Given two arrays, write a function to compute their intersection.

Notice

Each element in the result should appear as many times as it shows in both arrays.

The result can be in any order.

Example

Given `nums1 = [1, 2, 2, 1]`, `nums2 = [2, 2]`, return `[2, 2]`.

Challenge

What if the given array is already sorted? How would you optimize your algorithm?

用sort的方法。

What if `nums1`'s size is small compared to `num2`'s size? Which algorithm is better?

只对num1排序，然后用binary search。

What if elements of `nums2` are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

分段取

## Solution

和Intersection of Two Arrays类似，但是这题要求答案数组中要出现和原数组中相等数量的重复元素（两个数组中重复元素出现次数少的那个）。

第一种方法同样可以利用sort之后再比较，但是这次不用考虑答案数组中是否包含重复元素。

第二种方法可以利用HashMap，将两个数组中元素和重复次数分别存入两个HashMap，然后去两个HashMap中的重复元素和其中重复次数小的作为答案。

代码如下：

Sort

```
public class Solution {
    /**
     * @param nums1 an integer array
     * @param nums2 an integer array
     * @return an integer array
     */
    public int[] intersection(int[] nums1, int[] nums2) {
        // Write your code here
        Arrays.sort(nums1);
        Arrays.sort(nums2);

        ArrayList<Integer> res = new ArrayList<Integer>();
        int i = 0;
        int j = 0;
        while(i < nums1.length && j < nums2.length){
            if(nums1[i] == nums2[j]){
                res.add(nums1[i]);
                i++;
                j++;
            }else if(nums1[i] < nums2[j]){
                i++;
            }else{
                j++;
            }
        }

        int[] result = new int[res.size()];
        int count = 0;
        for(int item : res){
            result[count++] = item;
        }

        return result;
    }
}
```

## HashMap version1

```
public class Solution {
```

```
/**
 * @param nums1 an integer array
 * @param nums2 an integer array
 * @return an integer array
 */
public int[] intersection(int[] nums1, int[] nums2) {
    // Write your code here
    //hashmap version
    HashMap<Integer, Integer> map1 = new HashMap<Integer, Integer>();
    for(int i = 0; i < nums1.length; i++){
        if(!map1.containsKey(nums1[i])){
            map1.put(nums1[i], 1);
        }else{
            map1.put(nums1[i], map1.get(nums1[i]) + 1);
        }
    }

    HashMap<Integer, Integer> map2 = new HashMap<Integer, Integer>();
    for(int i = 0; i < nums2.length; i++){
        if(!map2.containsKey(nums2[i])){
            map2.put(nums2[i], 1);
        }else{
            map2.put(nums2[i], map2.get(nums2[i]) + 1);
        }
    }

    ArrayList<Integer> res = new ArrayList<Integer>();
    for(int key : map2.keySet()){
        if(map1.containsKey(key)){
            int count = Math.min(map1.get(key), map2.get(key));
            while(count != 0){
                res.add(key);
                count--;
            }
        }
    }
}
```

```
        int[] result = new int[res.size()];
        int count = 0;
        for(int i : res){
            result[count++] = i;
        }

        return result;
    }
}
```

HashMap version2

```
public class Solution {
    /**
     * @param nums1 an integer array
     * @param nums2 an integer array
     * @return an integer array
     */
    public int[] intersection(int[] nums1, int[] nums2) {
        // Write your code here
        Map<Integer, Integer> map = new HashMap<Integer, Integer>();
        for(int i = 0; i < nums1.length; ++i) {
            if (map.containsKey(nums1[i]))
                map.put(nums1[i], map.get(nums1[i]) + 1);
            else
                map.put(nums1[i], 1);
        }

        List<Integer> results = new ArrayList<Integer>();

        for (int i = 0; i < nums2.length; ++i)
            if (map.containsKey(nums2[i]) &&
                map.get(nums2[i]) > 0) {
                results.add(nums2[i]);
                map.put(nums2[i], map.get(nums2[i]) - 1);
            }

        int result[] = new int[results.size()];
        for(int i = 0; i < results.size(); ++i)
            result[i] = results.get(i);

        return result;
    }
}
```



# Minimum Window Substring 32

## Question

Given a string source and a string target, find the minimum window in source which will contain all the characters in target.

Notice

If there is no such window in source that covers all characters in target, return the empty string "".

If there are multiple such windows, you are guaranteed that there will always be only one unique minimum window in source.

Clarification

Should the characters in minimum window has the same order in target?

Not necessary.

Example

For source = "ADOBECODEBANC", target = "ABC", the minimum window is "BANC"

Challenge

Can you do it in time complexity  $O(n)$  ?

## Solution

1. 用一个数组Target（256长度，java中一共有256个字符）记录target中所有字符及其数量
2. 用sourceNum来记录source中和target中匹配上的字符数量，用两个指针left和right来表示当前子字符串的左右边界

3. 从左往右遍历source，每遇到一位字符，将Target中对应字符的数量-1，若Target中该字符的数量大于0，则表示该字符在target中，将sourceNum+1
4. 当sourceNum和target长度相等时表明target中字符全部被当前子字符串匹配上，此时比较之前匹配上的子字符串长度，若当前子字符串较短，则更新最短子字符串长度和内容
5. 将当前子字符串左边界向右移1位，Target中对应字符（即原左边界字符）数量增加1，若Target中该字符数量大于0，则表示和target中匹配上的字符-1，否则只是删去多余字符，不影响
6. 当sourceNum小于target长度时，表示当前子字符串已经无法完全匹配target中字符，向右移动右边界，重复3-5步骤，寻找新的合适的子字符串，直到source遍历完成

代码如下：

```
public class Solution {
    /**
     * @param source: A string
     * @param target: A string
     * @return: A string denote the minimum window
     *         Return "" if there is no such a string
     */
    public String minWindow(String source, String target) {
        // write your code
        int ans = Integer.MAX_VALUE;
        String minStr = "";

        //java一共有0-255个字符
        int[] Target = new int[256];

        //寻找target中有哪些字符并统计其个数，并返回target中字符长度
        int targetNum = initialTargetHash(Target, target);

        //sourceNum代表source中和target中匹配上的字符数，等于0的为target中没有的字符，大于0的表示target中有的字符的数量
        int sourceNum = 0;

        //left和right代表当前子字符串的左右边界
```

```

        int left = 0;
        int right = 0;

        for(; right < source.length(); right++){

            //Target中个数大于0的表示target中有的字符个数，若匹配上，则
            匹配数+1
            if(Target[source.charAt(right)] > 0){
                sourceNum++;
            }

            //source中每增加一个字符会使Target中对应字符的数量-1，若小
            于0则表示source中的当前子字符串含有多余的字符（和target相比）
            Target[source.charAt(right)]--;

            //当匹配上的字符数大于等于target长度时，表示target中字符全部
            被匹配上
            while(sourceNum >= targetNum){

                //和之前的子字符串窗口的结果比较，寻找最小的子字符串窗口
                if(ans > right - left + 1){
                    ans = right - left + 1;
                    minStr = source.substring(left, right + 1);
                }

                //将当前子字符串左边界向右移1位，Target中相应字符数量增
                加1，若Target中该字符数量大于0，则表示和target中匹配上的字符-1，否则只是
                删去多余字符，不影响
                Target[source.charAt(left)]++;
                if(Target[source.charAt(left)] > 0){
                    sourceNum--;
                }
                left++;
            }
        }

        return minStr;
    }

    private int initialTargetHash(int[] Target, String target){

```

```
        int targetNum = 0;
        for(int i = 0; i < target.length(); i++){
            Target[target.charAt(i)]++;
            targetNum++;
        }
        return targetNum;
    }
}
```

# Max Points on a Line 186

## Question

Given  $n$  points on a 2D plane, find the maximum number of points that lie on the same straight line.

Example

Given 4 points: (1,2), (3,6), (0,0), (1,3).

The maximum number is 3.

## Solution

寻找一个数组中在一条线上的点最多有多少个。遍历数组，寻找以当前点为交点的所有线所包含的点的最大值。求交点之后所有点和交点的斜率，用一个hashmap保存斜率和该斜率的线所包含的点的数量。其中有几点要注意：

1. 每个交点只要和其之后的点计算斜率，因为若最后答案包含之前的点和该交点，则必定已经在之前的点为交点时计算过
2. 为了防止剩下数组中只有一个点无法计算斜率（如数组最后一个点），在每个新交点开始的时候都首先在hashmap中保存以min value为key数量为1的答案来cover只有一个点的特殊情况
3. 之后的点中可能有和交点相同的点，因为重复的点在一条线上要算多个，因此要记录有多少个重复点
4. 考虑和y轴垂直的直线（此时斜率不可计算），用一个max value来作为该直线的key
5. 斜率计算要用 $0.0 + \text{slop}$ 。原因： $\text{slop}$ 可能结果为 $-0.0$ ，用 $0.0 + -0.0 = 0.0$ ，来防止发生 $0.0 \neq -0.0$ 的情况。注意， $\text{slop}$ 一定要转化为double计算，否则结果不对。
6. 最后求该交点的最大值答案时，要加上该交点的重复点

代码如下：

```
/**
 * Definition for a point.
 * class Point {
 *     int x;
 *     int y;
 *     Point() { x = 0; y = 0; }
 *     Point(int a, int b) { x = a; y = b; }
 * }
 */
public class Solution {
    /**
     * @param points an array of point
     * @return an integer
     */
    public int maxPoints(Point[] points) {
        // Write your code here
        if(points == null || points.length == 0){
            return 0;
        }

        int max = 1;
        HashMap<Double, Integer> slop = new HashMap<Double, Integer>();
        //求以i为交点的所有line所包含点的最大值
        for(int i = 0; i < points.length; i++){
            //每换一个交点，map要清空
            slop.clear();
            //防止points里只有一个点
            slop.put((double)Integer.MIN_VALUE, 1);

            int dup = 0;
            for(int j = i + 1; j < points.length; j++){
                //用dup来记录和交点重复的点（重复的点在一条线上要算多个）
                if(points[j].x == points[i].x && points[j].y ==
points[i].y){
                    dup++;
                    continue;
                }
            }
        }
    }
}
```

//若line垂直，则key为MAX\_VALUE；用0.0+slop原因：slop可能结果为-0.0，用0.0+-0.0=0.0，来防止发生0.0 != -0.0的情况。注意，slop一定要转化为double计算，否则结果不对。

```
double key = points[j].x - points[i].x == 0? (double)Integer.MAX_VALUE : 0.0 + (double) (points[j].y - points[i].y) / (points[j].x - points[i].x);
```

```
        if(slop.containsKey(key)){
            slop.put(key, slop.get(key) + 1);
        }else{
            slop.put(key, 2);
        }
    }

    for(int value : slop.values()){
        if(value + dup > max){
            max = value + dup;
        }
    }
    return max;
}
}
```

# Longest Substring Without Repeating Characters 384

## Question

Given a string, find the length of the longest substring without repeating characters.

Example

For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3.

For "bbbb" the longest substring is "b", with the length of 1.

Challenge

$O(n)$  time

## Solution

我的方法：

1. 用一个index记录无重复的子字符串的左边界，用一个hashmap记录每个字符最靠右的位置。
2. 遍历数组，若hashmap中没有该字符，则在hashmap中添加该字符和其位置，若有，则将左边界更新为该字符之前位置，并将该字符更新为当前位置。
3. 更新左边界时，左边界只能向右移动，不能往左移，因此要判断更新位置和原来左边界的大小（如abccad）。
4. 若当前子字符串长度（当前字符位置减去左边界）比之前max大，则更新max。

九章的方法类似。

时间 $O(n)$ ，空间 $O(n)$ 。



代码如下：

我的方法：

```
public class Solution {  
    /**  
     * @param s: a string  
     * @return: an integer  
     */  
    public int lengthOfLongestSubstring(String s) {  
        // write your code here  
        if(s == null || s.length() == 0){  
            return 0;  
        }  
  
        int max = 0;  
        HashMap<Character, Integer> elem = new HashMap<Character  
, Integer>();  
        int last = -1;  
        for(int i = 0; i < s.length(); i++){  
            if(!elem.containsKey(s.charAt(i))){  
                elem.put(s.charAt(i), i);  
            }else{  
                //左边界只能往右移，不能往左移  
                if(elem.get(s.charAt(i)) > last){  
                    last = elem.get(s.charAt(i));  
                }  
                elem.put(s.charAt(i), i);  
            }  
            max = i - last > max? i - last : max;  
        }  
  
        return max;  
    }  
}
```

# Substring with Concatenation of All Words 30(LeetCode)

## Question

You are given a string, *s*, and a list of words, *words*, that are all of the same length. Find all starting indices of substring(s) in *s* that is a concatenation of each word in *words* exactly once and without any intervening characters.

For example, given:

*s*: "barfoothefoobarman"

*words*: ["foo", "bar"]

You should return the indices: [0,9]. (order does not matter).

## Solution

这道题思想很简单，就是用HashMap记录*words*中的信息，再用two pointers遍历*s*看有没有substring是由*words*中的所有元素组成的（顺序没关系）。

1. 用HashMap记录所有的*words*中的元素及其出现的次数
2. 然后用双指针开始遍历string。用一个指针记录起始位置，另一个指针开始以*words*中元素的长度开始扫描，如果遇到一个不在HashMap中的子字符串或者该子字符串出现的次数大于其在*words*中出现的次数则暂停，说明以该起始位置为起点的子字符串不能由*words*中的所有元素组成。
3. 继续下一个起始位置的扫描，重复2。直到某个起始位置到*s*的结尾的长度比*words*中所有元素的长度和小时，停止。

代码如下：

```
public class Solution {  
    public List<Integer> findSubstring(String s, String[] words)  
    {
```

```
List<Integer> res = new ArrayList<Integer>();
if(s == null || words == null || words.length == 0){
    return res;
}

HashMap<String, Integer> map = new HashMap<String, Integer>();
for(String str : words){
    if(map.containsKey(str)){
        map.put(str, map.get(str) + 1);
    }else{
        map.put(str, 1);
    }
}

int l = words[0].length();
int len = l * words.length;
int left, right;
int count;
for(left = 0; s.length() - left >= len; left++){
    HashMap<String, Integer> mymap = new HashMap<String, Integer>(map);
    right = left;
    String curt = s.substring(right, right + 1);
    count = 0;
    while(mymap.containsKey(curt) && mymap.get(curt) > 0){
        mymap.put(curt, mymap.get(curt) - 1);
        count++;
        right += 1;
        if(right >= s.length() || right + 1 > s.length()){
            break;
        }
        curt = s.substring(right, right + 1);
    }

    if(count == words.length){
        res.add(left);
    }
}
```

```
        }  
        return res;  
    }  
}
```

## Group Anagrams 49 ( LeetCode )

### Question

Given an array of strings, group anagrams together.

For example, given: ["eat", "tea", "tan", "ate", "nat", "bat"],

Return:

```
[["ate", "eat", "tea"],
```

```
["nat", "tan"],
```

```
["bat"]]
```

Note: All inputs will be in lower-case.

### Solution

这道题看所给的字符串数组里面有多少个是同一个变形词变的。这道题同样使用HashMap来帮助存老值和新值，以及帮忙判断是否是变形词。

首先对每个string转换成char array然后排下序，HashMap里面的key存sort后的词，value存原始的词。然后如果这个排好序的词没在HashMap中出现过，那么就把这个sorted word和unsortedword put进HashMap里面。如果一个sorted word是在HashMap里面存在过的，则直接加入HashMap中对应的位置。最后将HashMap中的每一个value都加入result中即可。

代码如下：

```
public class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {
        List<List<String>> res = new ArrayList<List<String>>();
        if(strs == null || strs.length == 0){
            return res;
        }

        HashMap<String, ArrayList<String>> map = new HashMap<String, ArrayList<String>>();
        for(int i = 0; i < strs.length; i++){
            String curt = strs[i];
            char[] curtArray = curt.toCharArray();
            Arrays.sort(curtArray);
            String sorted = new String(curtArray);
            if(!map.containsKey(sorted)){
                map.put(sorted, new ArrayList<String>());
            }
            map.get(sorted).add(curt);
        }

        for(String key : map.keySet()){
            res.add(map.get(key));
        }

        return res;
    }
}
```

## 217. Contains Duplicate

### Question

Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

### Solution

这道题很简单，用一个HashSet记录之前的数，利用HashSet的add方法，如果往set里加重复的数会返回false来判断是否有重复的数出现。

代码如下：

```
public class Solution {
    public boolean containsDuplicate(int[] nums) {
        if (nums == null || nums.length <= 1) {
            return false;
        }

        HashSet<Integer> set = new HashSet<>();
        for (int i = 0; i < nums.length; i++) {
            if (!set.add(nums[i])) {
                return true;
            }
        }
        return false;
    }
}
```

## 219. Contains Duplicate II

### Question

Given an array of integers and an integer  $k$ , find out whether there are two distinct indices  $i$  and  $j$  in the array such that  $\text{nums}[i] = \text{nums}[j]$  and the difference between  $i$  and  $j$  is at most  $k$ .

### Solution

这道题和Contains Duplicate I思路一样，但是因为多了index的差的范围的限制，考虑sliding windows，即window每次向右移动一格，添加新遇到的元素，删除最左边的元素，这样就能保证index的差在一定范围内。其他和Contains Duplicate I一样。

代码如下：

```
public class Solution {
    public boolean containsNearbyDuplicate(int[] nums, int k) {
        if (nums == null || nums.length <= 1) {
            return false;
        }

        HashSet<Integer> set = new HashSet<>();
        for (int i = 0; i < nums.length; i++) {
            if (i > k) {
                set.remove(nums[i - k - 1]);
            }
            if (!set.add(nums[i])) {
                return true;
            }
        }

        return false;
    }
}
```





## 220. Contains Duplicate III

### Question

Given an array of integers, find out whether there are two distinct indices  $i$  and  $j$  in the array such that the difference between  $\text{nums}[i]$  and  $\text{nums}[j]$  is at most  $t$  and the difference between  $i$  and  $j$  is at most  $k$ .

### Solution

保证indexd的差在一定范围内，用sliding window. 保证window中两个元素的差  $\leq t$ ，考虑以下两种方法：

方法一：利用TreeSet的数据结构，使用.floor(n)方法返回小于n的元素中的最大者，如果不存在则返回null，使用.ceiling(n)方法返回大于n的元素中的最小者，如果不存在则返回null。然后看当前元素 $\text{nums}[i]$ 和这两个元素的差是否小于 $t$ ，只要有一个小于就返回true。这里要注意的是，两个int的数相加或者相减很可能overflow，因此要将其转化为long再做比较。

方法二：利用bucket sort的思想，将每个num放到不同的bucket中去，然后只要比较当前的bucket以及前后的bucket就可以了，具体推导方法如下：

如果： $|\text{nums}[i] - \text{nums}[j]| \leq t$  式a

等价： $|\text{nums}[i] / t - \text{nums}[j] / t| \leq 1$  式b

推出： $|\text{floor}(\text{nums}[i] / t) - \text{floor}(\text{nums}[j] / t)| \leq 1$  式c

等价： $\text{floor}(\text{nums}[j] / t) \in \{\text{floor}(\text{nums}[i] / t) - 1, \text{floor}(\text{nums}[i] / t), \text{floor}(\text{nums}[i] / t) + 1\}$  式d

那么对于每一个num,只要 $\text{key} = \text{num} / t$ ，然后搜寻(key-1, key, key + 1)是否在维护的最大size为k的map中，并且判断真实的value diff是否 $\leq t$ 就可以了。

在这里要注意两点：

1) 我们将通式 $key = num/t$ 变为 $key = num / \text{Math.max}(1,t)$ 是为了应对当 $t$ 为0时的情况，虽然这时的 $key$ 会和 $t=1$ 时相同，但是在后面判断真实的 $value\ diff$ 是否 $\leq t$ 时依然可以根据 $t$ 的真实值进行判断。

2) 因为要搜寻 $key-1, key+1$ ，因此当 $key$ 为边界情况时可能造成 $overflow$ ，因此要将 $map$ 中的 $key$ 转换成 $long$ 进行存储，同时因为涉及到要判断真实的 $value\ diff$ 是否 $\leq t$ ，这时也有可能出现 $overflow$ 的情况，因此 $value$ 也要转换成 $long$ 进行存储。

代码如下：

```
public class Solution {
    public boolean containsNearbyAlmostDuplicate(int[] nums, int
k, int t) {
        if (nums == null || nums.length <= 1) {
            return false;
        }

        if (k < 1 || t < 0) {
            return false;
        }

        //TreeSet version
        // long diff = t;
        // TreeSet<Integer> set = new TreeSet<>();
        // for (int i = 0; i < nums.length; i++) {
        //     int curt = nums[i];
        //     if ((set.floor(curt) != null && curt <= diff + se
t.floor(curt)) || (set.ceiling(curt) != null && set.ceiling(curt
) <= diff + curt)) {
        //         return true;
        //     }
        //     set.add(curt);
        //     if (i >= k) {
        //         set.remove(nums[i - k]);
        //     }
        // }

        //Bucket version
        HashMap<Long, Long> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
```

```
        if (i > k) {
            int removeElement = nums[i - k - 1];
            long key = getId(removeElement, t);
            map.remove(key);
        }

        int curt = nums[i];
        long curtKey = getId(curt, t);
        if (map.containsKey(curtKey) && Math.abs(map.get(curtKey) - nums[i]) <= t) {
            return true;
        }
        if (map.containsKey(curtKey - 1) && Math.abs(map.get(curtKey - 1) - nums[i]) <= t) {
            return true;
        }
        if (map.containsKey(curtKey + 1) && Math.abs(map.get(curtKey + 1) - nums[i]) <= t) {
            return true;
        }
        map.put(curtKey, (long)curt);
    }

    return false;
}

private long getId(int n, int t) {
    return (long)n / Math.max(1, t);
}
}
```

# Deque

# Sliding Window Maximum 362

## Question

Given an array of  $n$  integer with duplicate number, and a moving window(size  $k$ ), move the window at each iteration from the start of the array, find the maximum number inside the window at each moving.

Example

For array  $[1, 2, 7, 7, 8]$ , moving window size  $k = 3$ . return  $[7, 7, 8]$

At first the window is at the start of the array like this

$[1, 2, 7 | 7, 8]$  , return the maximum 7;

then the window move one step forward.

$[1, | 2, 7, 7 |, 8]$ , return the maximum 7;

then the window move one step forward again.

$[1, 2, | 7, 7, 8 |]$ , return the maximum 8;

Challenge

$O(n)$  time and  $O(k)$  memory

## Solution

用Deque双端队列来解。分为添加元素和删除元素两部步。

1. 初始化第一个窗口（0-第 $k-1$ 个元素），一次向deque中加入数组中的元素。每次加入新元素时，若deque中最后一个元素比新元素小，则删去，直到deque中最后一个元素比新元素大时停止（或deque为空时停止），然后加入新元素。

2. 添加元素：从第 $k$ 的元素开始，每次加入新元素时，若deque中最后一个元素比新元素小，则删去，直到deque中最后一个元素比新元素大时停止（或deque为空时停止），然后加入新元素。此时deque中第一个元素即为当前窗口的最大值，加入答案中。
3. 删除元素：应该删去（当前位置 $-k$ ）位置的元素，看deque第一个元素是否和要删除元素相等，若不相等则说明在之前的元素加入过程中该元素已经删去，则不用做任何操作，否则删去deque中第一个元素即可。

代码如下：

```
public class Solution {
    /**
     * @param nums: A list of integers.
     * @return: The maximum number inside the window at each moving.
     */
    public ArrayList<Integer> maxSlidingWindow(int[] nums, int k) {
        // write your code here
        ArrayList<Integer> res = new ArrayList<Integer>();
        if(nums == null || nums.length < k || k <= 0){
            return res;
        }

        Deque<Integer> deque = new ArrayDeque<Integer>();
        for(int i = 0; i < k - 1; i++){
            while(!deque.isEmpty() && deque.getLast() < nums[i])
            {
                deque.removeLast();
            }
            deque.offer(nums[i]);
        }
        for(int i = k - 1; i < nums.length; i++){
            while(!deque.isEmpty() && deque.getLast() < nums[i])
            {
                deque.removeLast();
            }
            deque.offer(nums[i]);

            res.add(deque.getFirst());

            if(deque.getFirst() == nums[i - k + 1]){
                deque.removeFirst();
            }
        }

        return res;
    }
}
```





# Matrix

# Matrix Zigzag Traversal 185

## Question

Given a matrix of  $m \times n$  elements ( $m$  rows,  $n$  columns), return all elements of the matrix in ZigZag-order.

Example

Given a matrix:

[ [1, 2, 3, 4], [5, 6, 7, 8], [9,10, 11, 12] ]

return [1, 2, 5, 9, 6, 3, 4, 7, 10, 11, 8, 12]

## Solution

沿着斜线往上走，往下走，再往上走。。。直到最后一个元素。用 $dx$ ， $dy$ 来控制走的方向，每次走到上下边界之后就换方向。

代码如下：

```
public class Solution {
    /**
     * @param matrix: a matrix of integers
     * @return: an array of integers
     */
    public int[] printZMatrix(int[][] matrix) {
        // write your code here
        if(matrix == null || matrix.length == 0 || matrix[0].length == 0){
            return new int[0];
        }

        int m = matrix.length;
        int n = matrix[0].length;
        int[] res = new int[m * n];
```

```
res[0] = matrix[0][0];
int count = 1;
int x = 0;
int y = 0;
int dx = -1;
int dy = 1;

while(count < m * n){
    if(x + dx >= 0 && y + dy >= 0 && x + dx < m && y + d
y < n){

        //如果还没走到头，则继续往斜上方或者斜下方走
        x = x + dx;
        y = y + dy;
    }else{
        //往上走走到头
        if(dx == -1 && dy == 1){
            //如果还能向右移动则向右移，不行就向下移
            if(y + dy < n){
                y++;
            }else{
                x++;
            }
            dx = 1;
            dy = -1;
        }else{//往下走走到头
            //如果还能向下移动则向下移，否则向右移
            if(x + dx < m){
                x++;
            }else{
                y++;
            }
            dx = -1;
            dy = 1;
        }
    }
    res[count++] = matrix[x][y];
}

return res;
}
```

}

# Valid Sudoku 389

## Question

Determine whether a Sudoku is valid.

The Sudoku board could be partially filled, where empty cells are filled with the character ..

Notice

A valid Sudoku board (partially filled) is not necessarily solvable. Only the filled cells need to be validated.

Example

The following partially filed sudoku is valid.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

## Solution

分别检查行，列，小矩阵是否合法。用一个array来记录1-9是否被访问，每次检查新的之前要清空。（也可以用hashset来保存，不过array效率更高）。小矩阵每个cell的index也要注意。

代码如下：

```
class Solution {
```

```
/**
 * @param board: the board
 * @return: whether the Sudoku is valid
 */
public boolean isValidSudoku(char[][] board) {
    if(board == null || board.length == 0 || board[0].length
== 0){
        return false;
    }

    boolean[] visited = new boolean[9];

    //row
    for(int i = 0; i < 9; i++){
        Arrays.fill(visited, false);
        for(int j = 0; j < 9; j++){
            if(!isValid(visited, board[i][j])){
                return false;
            }
        }
    }

    //column
    for(int i = 0; i < 9; i++){
        Arrays.fill(visited, false);
        for(int j = 0; j < 9; j++){
            if(!isValid(visited, board[j][i])){
                return false;
            }
        }
    }

    //submatrix
    for(int i = 0; i < 9; i += 3){
        for(int j = 0; j < 9; j += 3){
            Arrays.fill(visited, false);
            for(int k = 0; k < 9; k++){
                if(!isValid(visited, board[i + k / 3][j + k
% 3])){
                    return false;
                }
            }
        }
    }
}
```

```
        }
    }
}

return true;
}

private boolean isValid(boolean[] visited, char c){
    //'.'
    if(c == '.'){
        return true;
    }

    int index = c - '0';
    if(index < 1 || index > 9 || visited[index - 1]){
        return false;
    }

    visited[index - 1] = true;
    return true;
}
};
```



# Rotate Image 161

## Question

You are given an  $n \times n$  2D matrix representing an image. Rotate the image by 90 degrees (clockwise).

Example

Given a matrix

```
[[1,2], [3,4]]
```

rotate it by 90 degrees (clockwise), return

```
[[3,1], [4,2]]
```

Challenge

Do it in-place.

## Solution

思想很简单，就是左上和右上换，右上和右下换，右下和左下换，左下和左上换。每次从4个角开始换起，用count记录从matrix往里的步数。index可能稍微麻烦一些，需要记住上下是行不变列变，左右是列不变行变，不变的和count相关，变化的和i相关。

代码如下：

```
public class Solution {
    /**
     * @param matrix: A list of lists of integers
     * @return: Void
     */
    public void rotate(int[][] matrix) {
        // write your code here
        if(matrix == null || matrix.length == 0){
            return;
        }

        int n = matrix.length;
        int count = 0;
        while(count < n - 1 - count){
            for(int i = count; i < n - count - 1; i++){
                int temp = matrix[count][i];
                matrix[count][i] = matrix[n - 1 - i][count];
                matrix[n - 1 - i][count] = matrix[n - 1 - count]
[n - 1 - i];
                matrix[n - 1 - count][n - 1 - i] = matrix[i][n -
1 - count];
                matrix[i][n - 1 - count] = temp;
            }
            count++;
        }
    }
}
```

## Search a 2D Matrix II 38

### Question

Write an efficient algorithm that searches for a value in an  $m \times n$  matrix, return the occurrence of it.

This matrix has the following properties:

Integers in each row are sorted from left to right.

Integers in each column are sorted from up to bottom.

No duplicate integers in each row or column.

Example

Consider the following matrix:

`[[1, 3, 5, 7], [2, 4, 7, 8], [3, 5, 9, 10]]`

Given target = 3, return 2.

Challenge

$O(m+n)$  time and  $O(1)$  extra space

### Solution

从左下角开始向右上角搜索，若当前元素比target小则向右，大则向上，相等则向右上。这样最多走 $m+n$ 步，所以时间复杂度为 $O(m+n)$ 。

代码如下：

```
public class Solution {  
    /**  
     * @param matrix: A list of lists of integers  
     * @param: A number you want to search in the matrix  
     * @return: An integer indicate the occurrence of target in  
the given matrix  
     */  
    public int searchMatrix(int[][] matrix, int target) {  
        // write your code here  
        if(matrix == null || matrix.length == 0 || matrix[0].length == 0){  
            return 0;  
        }  
  
        int x = matrix.length - 1;  
        int y = 0;  
        int count = 0;  
  
        while(x >= 0 && y < matrix[0].length){  
            if(matrix[x][y] == target){  
                count++;  
                x--;  
                y++;  
            }else if(matrix[x][y] < target){  
                y++;  
            }else{  
                x--;  
            }  
        }  
  
        return count;  
    }  
}
```

# Set Matrix Zeroes 162

## Question

Given a  $m \times n$  matrix, if an element is 0, set its entire row and column to 0. Do it in place.

Example

Given a matrix

`[ [1,2], [0,3] ],`

return

`[ [0,2], [0,0] ]`

Challenge

Did you use extra space?

A straight forward solution using  $O(mn)$  space is probably a bad idea.

A simple improvement uses  $O(m + n)$  space, but still not the best solution.

Could you devise a constant space solution?

## Solution

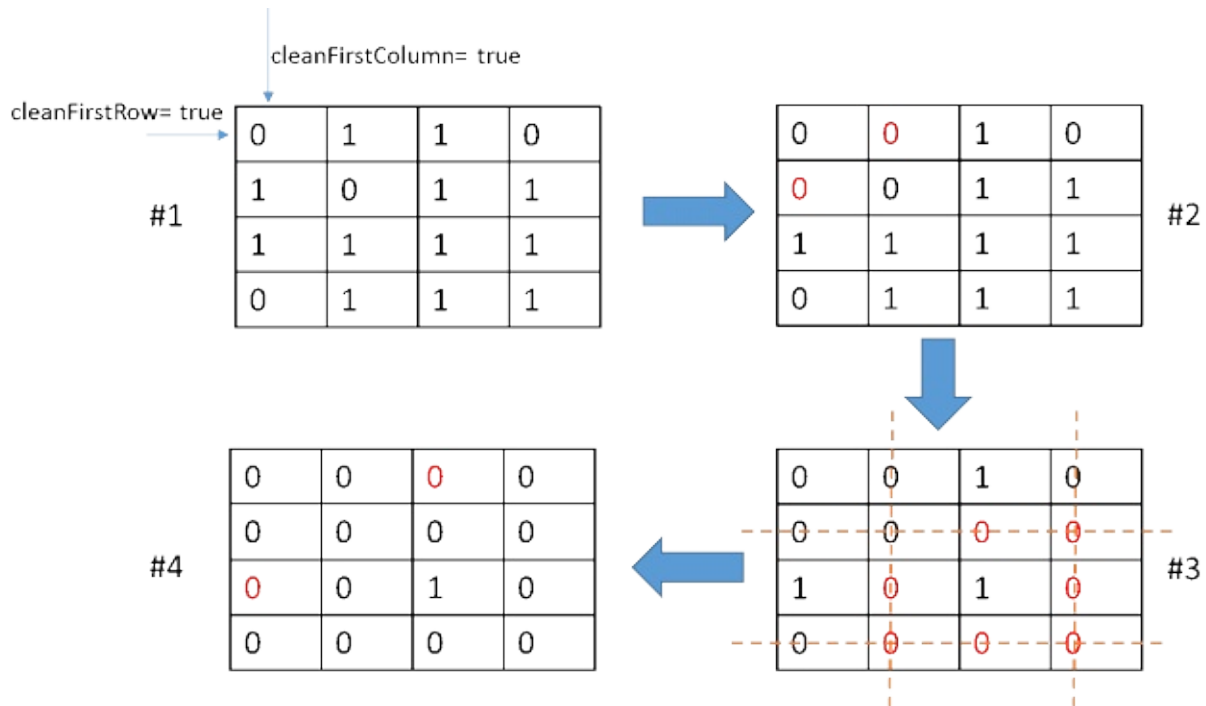
最蠢的方法是记录所有元素的状态，看是原来是0还是被变成的0，这样需要空间  $O(mn)$ 。

然后可以用一个数组来记录要变成0的行和列的位置，这样最多有  $m$  行  $n$  列，需要空间  $O(m + 1)$ 。

这道题的要求是服用空间，即用 `matrix` 的第一行和第一列来存储要变成0的标志的位置。

1. 先确定第一行和第一列是否需要清零

2. 扫描剩下的矩阵元素，如果遇到了0，就将对应的第一行和第一列上的元素赋值为0
3. 根据第一行和第一列的信息，已经可以讲剩下的矩阵元素赋值为结果所需的值了
4. 根据1中确定的状态，处理第一行和第一列



代码如下：

```
public class Solution {
    /**
     * @param matrix: A list of lists of integers
     * @return: Void
     */
    public void setZeroes(int[][] matrix) {
        // write your code here
        if(matrix == null || matrix.length == 0 || matrix[0].length == 0){
            return;
        }

        //确定第一行是否要变成0
        boolean rowZero = false;
        for(int j = 0; j < matrix[0].length; j++){
```

```
        if(matrix[0][j] == 0){
            rowZero = true;
            break;
        }
    }
    //确定第一列是否要变成0
    boolean columnZero = false;
    for(int i = 0; i < matrix.length; i++){
        if(matrix[i][0] == 0){
            columnZero = true;
            break;
        }
    }
    //扫描剩下元素，若遇到0，则将其所在行和列的第一个元素变为0
    for(int i = 1; i < matrix.length; i++){
        for(int j = 1; j < matrix[0].length; j++){
            if(matrix[i][j] == 0){
                matrix[0][j] = 0;
                matrix[i][0] = 0;
            }
        }
    }
    //根据第一列信息，将所有要变0的行都变为0
    for(int i = 1; i < matrix.length; i++){
        if(matrix[i][0] == 0){
            for(int j = 1; j < matrix[0].length; j++){
                matrix[i][j] = 0;
            }
        }
    }
    //根据第一行信息，将所有要变0的列都变为0
    for(int j = 1; j < matrix[0].length; j++){
        if(matrix[0][j] == 0){
            for(int i = 1; i < matrix.length; i++){
                matrix[i][j] = 0;
            }
        }
    }
    //根据一开始的信息，将第一列变为0
    if(columnZero){
```

```
        for(int i = 0; i < matrix.length; i++){
            matrix[i][0] = 0;
        }
    }
    //根据一开始的信息，将第一行变为0
    if(rowZero){
        for(int j = 0; j < matrix[0].length; j++){
            matrix[0][j] = 0;
        }
    }
}
```



# Sliding Window Matrix Maximum 558

## Question

Given an array of  $n \times m$  matrix, and a moving matrix window (size  $k \times k$ ), move the window from top left to bottom right at each iteration, find the maximum sum of the elements inside the window at each moving. Return 0 if the answer does not exist.

Example

For matrix

[ [1, 5, 3], [3, 2, 1], [4, 1, 9], ]

The moving window size  $k = 2$ .

return 13.

At first the window is at the start of the array like this

[ [1, 5], 3], [3, 2], 1], [4, 1, 9], ]

,get the sum 11;

then the window move one step forward.

[ [1, 5, 3], [3, 2], 1], [4, 1, 9], ]

,get the sum 11;

then the window move one step forward again.

[ [1, 5, 3], [3, 2], 1], [4, 1], 9], ]

,get the sum 10;

then the window move one step forward again.

[ [1, 5, 3], [3, 2], 1], [4, 1, 9], ] ,get the sum 13;

SO finally, get the maximum from all the sum which is 13.

## Challenge

$O(n^2)$  time.

## Solution

$sum[i][j]$  表示  $0-i-1$  行和  $0-j-1$  列所有元素的和。

1. 建立  $sum$  矩阵，为  $n+1$  行， $m+1$  列。将第 0 行和第 0 列都初始化为 0。
2. 遍历  $matrix$ ，根据公式  $sum[i][j] = matrix[i-1][j-1] + sum[i][j-1] + sum[i-1][j] - sum[i-1][j-1]$  计算所有  $sum$ 。
3. 然后计算每个窗口的  $sum$ ，根据公式  $sum = sum[i][j] - sum[i-k][j] - sum[i][j-k] + sum[i-k][j-k]$  即整个大窗口剪去上面一部分和左边一部分再加回被重复减去的部分，即为当前窗口值

代码如下：

```
public class Solution {
    /**
     * @param matrix an integer array of n * m matrix
     * @param k an integer
     * @return the maximum number
     */
    public int maxSlidingWindow2(int[][] matrix, int k) {
        // Write your code here
        if(matrix == null || matrix.length == 0 || matrix[0].length == 0 || k > matrix.length || k > matrix[0].length){
            return 0;
        }

        int n = matrix.length;
        int m = matrix[0].length;
        int[][] sum = new int[n + 1][m + 1];

        for(int i = 0; i <= n; i++){
            sum[i][0] = 0;
        }
    }
}
```

```
        for(int j = 1; j <= m; j++){
            sum[0][j] = 0;
        }

        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= m; j++){
                sum[i][j] = matrix[i - 1][j - 1] + sum[i][j - 1]
+ sum[i - 1][j] -sum[i - 1][j - 1];
            }
        }

        int max = Integer.MIN_VALUE;
        for(int i = k; i <= n; i++){
            for(int j = k; j <= m; j++){
                int temp = sum[i][j] - sum[i - k][j] - sum[i][j
- k] + sum[i - k][j - k];
                max = Math.max(max, temp);
            }
        }

        return max;
    }
}
```

# Submatrix Sum 405

## Question

Given an integer matrix, find a submatrix where the sum of numbers is zero. Your code should return the coordinate of the left-up and right-down number.

Example

Given matrix

```
[ [1 ,5 ,7], [3 ,7 ,-8], [4 ,-8 ,9], ]
```

```
return [(1,1), (2,2)]
```

Challenge

$O(n^3)$  time.

## Solution

这道题和求数组中哪些元素和为0的解决方法一样，只是数组中求的是前*i*个元素和前*j*个元素和相等，则*i-j*元素和为0，而这里只是变成2维的而已。

$sum[i][j]$ 表示 $matrix[0][0]$ 到 $matrix[i-1][j-1]$ 所有元素的和。

1. 建立sum矩阵，为*n*+1行，*m*+1列。将第0行和第0列都初始化为0。
2. 遍历matrix，根据公式  $sum[i][j] = matrix[i-1][j-1] + sum[i][j-1] + sum[i-1][j] - sum[i-1][j-1]$  计算所有sum。
3. 然后取两个row：l1, l2。用一个线k从左到右扫过l1和l2，每次都使用 $diff = sum[l1][k] - sum[l2][k]$ 来表示l1-l2和0-k这个矩形元素的sum。如果在同一个l1和l2中，有两条线（k1, k2）的diff相等，则表示l1-l2和k1-k2这个矩形中的元素和为0。

代码如下：

```
public class Solution {  
    /**
```

```

    * @param matrix an integer matrix
    * @return the coordinate of the left-up and right-down number
    */
    public int[][] submatrixSum(int[][] matrix) {
        // Write your code here
        int[][] res = new int[2][2];
        if(matrix == null || matrix.length == 0 || matrix[0].length == 0){
            return res;
        }

        int m = matrix.length;
        int n = matrix[0].length;
        int[][] sum = new int[m + 1][n + 1];

        for(int i = 0; i < m; i++){
            sum[i][0] = 0;
        }

        for(int j = 0; j < n; j++){
            sum[0][j] = 0;
        }

        for(int i = 1; i <= m; i++){
            for(int j = 1; j <= n; j++){
                sum[i][j] = matrix[i - 1][j - 1] + sum[i - 1][j]
+ sum[i][j - 1] - sum[i - 1][j - 1];
            }
        }

        for(int l = 0; l < m; l++){
            for(int h = l + 1; h <= m; h++){
                HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
                for(int k = 0; k <= n; k++){
                    int diff = sum[h][k] - sum[l][k];
                    if(map.containsKey(diff)){
                        /////在matrix中和sum对应的点的index都要-1，在matrix中，左上角的点应该是循环中的点的右下方的点：l+1-1,map.get(diff)+1-1 (

```

即不包含循环中的左上角的点)，而右下角的点是包含循环中右下角的点： $h-1, k-1$

```
        res[0][0] = 1;
        res[0][1] = map.get(diff);
        res[1][0] = h - 1;
        res[1][1] = k - 1;
        return res;
    }else{
        map.put(diff, k);
    }
    }
    }
    return res;
}
}
```

## Find Peak Element II 390

### Question

There is an integer matrix which has the following features:

The numbers in adjacent positions are different.

The matrix has  $n$  rows and  $m$  columns.

For all  $i < m$ ,  $A[0][i] < A[1][i]$  &&  $A[n-2][i] > A[n-1][i]$ .

For all  $j < n$ ,  $A[j][0] < A[j][1]$  &&  $A[j][m-2] > A[j][m-1]$ .

We define a position  $P$  is a peak if:

$$A[j][i] > A[j+1][i] \text{ \&\& } A[j][i] > A[j-1][i] \text{ \&\& } A[j][i] > A[j][i+1] \text{ \&\& } A[j][i] > A[j][i-1]$$

Find a peak element in this matrix. Return the index of the peak.

Example

Given a matrix:

```
[ [1, 2, 3, 6, 5], [16, 41, 23, 22, 6], [15, 17, 24, 21, 7], [14, 18, 19, 20, 10], [13, 14, 11, 10, 9] ]
```

return index of 41 (which is [1,1]) or index of 24 (which is [2,2])

Challenge

Solve it in  $O(n+m)$  time.

If you come up with an algorithm that you thought it is  $O(n \log m)$  or  $O(m \log n)$ , can you prove it is actually  $O(n+m)$  or propose a similar but  $O(n+m)$  algorithm?

### Solution

和在数组中find peak element一样，对行和列分别进行二分查找。

1. 先对行进行二分搜索，对搜到的那一行元素再进行二分搜索寻找peak element

2. 对找到的element看上下行的同列元素，若相同则返回，若比上小则在上半部分行继续进行搜索，若比下小则在下半部分的行继续进行搜索

代码如下：

```
class Solution {
    /**
     * @param A: An integer matrix
     * @return: The index of the peak
     */
    public List<Integer> findPeakII(int[][] A) {
        // write your code here
        List<Integer> res = new ArrayList<Integer>();
        if(A == null || A.length == 0 || A[0].length == 0){
            return res;
        }
        //根据题意，第1行和最后一行都不可能是peak，所以从第2行和倒数第2行
        开始

        int low = 1;
        int high = A.length - 2;

        while(low <= high){
            int mid = low + (high - low) / 2;
            int col = findPeak(mid, A);
            if(A[mid][col] < A[mid - 1][col]){
                high = mid - 1;
            }else if(A[mid][col] < A[mid + 1][col]){
                low = mid + 1;
            }else{
                res.add(mid);
                res.add(col);
                break;
            }
        }

        return res;
    }

    private int findPeak(int row, int[][] A){
```



```
int start = 0;
int end = A[row].length - 1;

while(start + 1 < end){
    int mid = start + (end - start) / 2;
    if(A[row][mid] > A[row][mid - 1] && A[row][mid] > A[
row][mid + 1]){
        return mid;
    }else if(A[row][mid] < A[row][mid - 1]){
        end = mid;
    }else{
        start = mid;
    }
}

if(A[row][start] > A[row][end]){
    return start;
}else{
    return end;
}
}
```

## Sudoku Solver 37 (LeetCode)

### Question

Write a program to solve a Sudoku puzzle by filling the empty cells.

Empty cells are indicated by the character '.'.

You may assume that there will be only one unique solution.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

A sudoku puzzle...

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

...and its solution numbers marked in red.

### Solution

看到这道题应该就能想到是尝试在一个格子填可能的数，检查是否valid再递归填剩下的格子。因此自然想到用backtrack，其中check valid步骤和Valid Sudoku一样，这里不再赘述。

遍历Sudoku中的每一个格子，如果已经有数字就继续下一个，如果没有就在里面填数字i（1~9），检查填了之后是否valid并且递归寻找新board中（即填了i之后的board）剩余格子能否有可行解，如果两个条件都满足，则直接返回true表示这个格子填数字i有可行解，否则将该格子重置为'.'，并继续尝试下一个数字。

代码如下：

```
public class Solution {
    public void solveSudoku(char[][] board) {
        solve(board, 0, 0);
    }

    private boolean solve(char[][] board, int row, int col){
        if(col == board[0].length){
            row += 1;
            col = 0;
        }

        for(int i = 0; i < board.length; i++){
            for(int j = 0; j < board[0].length; j++){
                if(board[i][j] != '.'){
                    continue;
                }

                for(int k = 1; k <= 9; k++){
                    board[i][j] = (char) ('0' + k);
                    if(isValid(board, i, j) && solve(board, i, j
+ 1)){
                        return true;
                    }
                    board[i][j] = '.';
                }
                return false;
            }
        }
        return true;
    }
}
```

```
}

private boolean isValid(char[][] board, int row, int col){
    HashSet<Character> set = new HashSet<Character>();
    for(int i = 0; i < board.length; i++){
        if(set.contains(board[i][col])){
            return false;
        }
        if(board[i][col] != '.'){
            set.add(board[i][col]);
        }
    }

    set = new HashSet<Character>();
    for(int i = 0; i < board[0].length; i++){
        if(set.contains(board[row][i])){
            return false;
        }
        if(board[row][i] != '.'){
            set.add(board[row][i]);
        }
    }

    set = new HashSet<Character>();
    int r = (row / 3) * 3;
    int c = (col / 3) * 3;
    for(int i = r; i < r + 3; i++){
        for(int j = c; j < c + 3; j++){
            if(set.contains(board[i][j])){
                return false;
            }
            if(board[i][j] != '.'){
                set.add(board[i][j]);
            }
        }
    }

    return true;
}
}
```

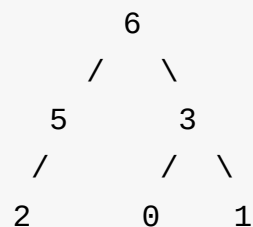


# Max Tree 126

## Question

Given an integer array with no duplicates. A max tree building on this array is defined as follow: 1) The root is the maximum number in the array 2) The left subtree and right subtree are the max trees of the subarray divided by the root number. Construct the max tree by the given array.

Example: Given [2, 5, 6, 0, 3, 1], the max tree is



## Solution

这题要我们用一个数组构建一个Max Tree，其定义与Heap类似，都要求每个节点必须比其子节点大，但是和Heap不同的地方在于Max Tree不要求左边先填满。主要思想为利用单调递减栈，即栈中每个元素都比其右边的元素大。具体方法如下：

- 1) 如果新来一个数，比堆栈顶的树根的数小，则把这个数作为一个单独的节点压入堆栈。
- 2) 否则，不断从堆栈里弹出树，新弹出的树以旧弹出的树为右子树，连接起来，直到目前堆栈顶的树根的数大于新来的数。然后，弹出的那些数，已经形成了一个新的树，这个树作为新节点的左子树，最后把这个新树压入堆栈。

这样的堆栈是单调递减的，越靠近堆栈顶的数越小。栈中存储的都是子树的根，且都只有左子树没有右子树。最后还要按照（2）的方法，把所有树弹出来，每个旧树作为新树的右子树。

代码如下：

\*Definition of TreeNode:

```
* public class TreeNode {
*     public int val;
*     public TreeNode left, right;
*     public TreeNode(int val) {
*         this.val = val;
*         this.left = this.right = null;
*     }
* }
```

```
public class Solution {
```

```
    /**
```

```
     * @param A: Given an integer array with no duplicates.
```

```
     * @return: The root of max tree.
```

```
    */
```

```
    public TreeNode maxTree(int[] A) {
```

```
        if (A.length==0) return null;
```

```
        Stack<TreeNode> nodeStack = new Stack<TreeNode>();
```

```
        nodeStack.push(new TreeNode(A[0]));
```

```
        for (int i=1;i<A.length;i++)
```

```
            if (A[i]<=nodeStack.peek().val){
```

```
                TreeNode node = new TreeNode(A[i]);
```

```
                nodeStack.push(node);
```

```
            } else {
```

```
                TreeNode n1 = nodeStack.pop();
```

```
                while (!nodeStack.isEmpty() && nodeStack.peek().
```

```
val < A[i]){
```

```
                    TreeNode n2 = nodeStack.pop();
```

```
                    n2.right = n1;
```

```
                    n1 = n2;
```

```
                }
```

```
                TreeNode node = new TreeNode(A[i]);
```

```
                node.left = n1;
```

```
                nodeStack.push(node);
```

```
            }
```

```
        TreeNode root = nodeStack.pop();
```

```
        while (!nodeStack.isEmpty()){
```

```
        nodeStack.peek().right = root;
        root = nodeStack.pop();
    }

    return root;
}
```



# Top K Frequent Words 471

## Question

Given a list of words and an integer k, return the top k frequent words in the list.

Example:

```
[ "yes", "lint", "code",  
  "yes", "code", "baby",  
  "you", "baby", "chrome",  
  "safari", "lint", "code",  
  "body", "lint", "code" ]
```

当k=3时，返回["code", "lint", "baby"]

当k=4时，返回["code", "lint", "baby", "yes"]

Note

You should order the words by the frequency of them in the return list, the most frequent one comes first. **If two words has the same frequency, the one with lower alphabetical order come first.**

Challenge

Do it in  $O(n \log k)$  time and  $O(n)$  extra space.

Extra points if you can do it in  $O(n)$  time with  $O(k)$  extra space.

Tags Expand

Hash Table Heap Priority Queue

## Discussion

1) 统计每个单词出现次数

- 2) 用最大堆来找到次数最大的单词
- 3) 用最小堆来确定相同次数时单词的顺序

## Solution

### 1) HashMap + PriorityQueue(MinHeap)

遍历数组来统计词频。用一个Node class来表示每个单词和其词频。用一个HashMap来存每个单词和其node。用一个PriorityQueue来对HashMap中的node进行排序，重构Comparator来实现1)按词频重大到小排序2)词频相同时，按字典序排序。然后依次取k个queue顶元素。

### 2) Trie + MinHeap

学完Trietree再来补上

代码如下：

HashMap + PriorityQueue:

```
public class Solution {
    class Node{
        int frequency;
        String str;
        public Node(String s, int f){
            frequency = f;
            str = s;
        }
    }

    public String[] topKFrequentWords(String[] string, int k){

        if(string == null || string.length == 0 || k <= 0){
            return null;
        }

        HashMap<String, Node> map = new HashMap<String, Node>();
        for(String curt : string){
            if(!map.containsKey(curt)){
```

```
        map.put(curt, new Node(curt, 1));
    }
    map.get(curt).frequency = map.get(curt).frequency +
1;
    }

    PriorityQueue<Node> queue = new PriorityQueue<Node>(k, new Comparator<Node>(){
        public int compare(Node n1, Node n2){
            if(n1.frequency == n2.frequency){
                return n1.str.compareTo(n2.str);
            }else{
                return n2.frequency - n1.frequency;
            }
        }
    });

    for(String key : map.keySet()){
        queue.offer(map.get(key));
    }

    String[] result = new String[k];
    for(int i = 0; i < k; i++){
        result[i] = queue.poll().str;
    }

    return result;
}
}
```

Trie + MinHeap:

# Implement Stack 495

## Question

实现一个栈，可以使用除了栈之外的数据结构。实现栈的基本功能：push, pop, top, isEmpty。

Example:

```
push(1)
```

```
pop()
```

```
push(2)
```

```
top() // return 2
```

```
pop()
```

```
isEmpty() // return true
```

```
push(3)
```

```
isEmpty() // return false
```

## Solution

使用LinkedList实现stack。

代码如下：

```
class ListNode{
    int val;
    ListNode next;
    public ListNode(int val){
        this.val = val;
    }
}

class Stack{
    ListNode head;
    public Stack(){
        head = new ListNode(0);
    }

    public void push(int value){
        ListNode node = new ListNode(value);
        node.next = head.next;
        head.next = node;
    }

    public int pop(){
        ListNode top = head.next;
        head.next = head.next.next;
        return top.val;
    }

    public int top(){
        return head.next.val;
    }

    public boolean isEmpty(){
        if(head.next == null){
            return true;
        }else{
            return false;
        }
    }
}
```

## Related Question

[Implement stack by two queues 494](#)

# Implement Stack by Two Queues 494

## Question

Implement the following operations of a stack using queues.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- empty() -- Return whether the stack is empty.

Notes:

- You may assume that all operations are valid (for example, no pop or top operations will be called on an empty stack).
- Depending on your language, queue may not be supported natively. You may simulate a queue by using a list or deque (double-ended queue), as long as you use only standard operations of a queue -- which means only push to back, pop from front, size, and is empty operations are valid.

## Solution

水题。同Implement queue by two stacks。

代码如下：

```
class Stack{
    private Queue<Integer> queue1;
    private Queue<Integer> queue2;

    public Stack(){
        queue1 = new LinkedList<Integer>();
        queue2 = new LinkedList<Integer>();
    }
}
```

```
public void push(int value){
    queue1.offer(value);
}

public int pop(){
    while(queue1.size() != 1){
        queue2.offer(queue1.poll());
    }
    int res = queue1.poll();
    while(!queue2.isEmpty()){
        queue1.offer(queue2.poll());
    }
    return res;
}

public int top(){
    while(queue1.size() != 1){
        queue2.offer(queue1.poll());
    }
    int res = queue1.poll();
    queue2.offer(res);
    while(!queue2.isEmpty()){
        queue1.offer(queue2.poll());
    }
    return res;
}

public boolean isEmpty(){
    return queue1.isEmpty();
}
}
```



# Implement Queue by Linked List II 493

## Question

Implement a Queue by linked list. Provide the following basic methods:

`push_front(item)`. Add a new item to the front of queue.

`push_back(item)`. Add a new item to the back of the queue.

`pop_front()`. Move the first item out of the queue, return it.

`pop_back()`. Move the last item out of the queue, return it.

Example:

```
push_front(1)
```

```
push_back(2)
```

```
pop_back() // return 2
```

```
pop_back() // return 1
```

```
push_back(3)
```

```
push_back(4)
```

```
pop_front() // return 3
```

```
pop_front() // return 4
```

## Solution

用`head`和`tail`记录队列的头尾。两点注意：

1) 要在`front`插入，所以`head`为dummy node（其`next`为队头node），`tail`为队尾node。

2) 要在`back`出队，所以用双向链表，这样队尾node出队后`tail`可以往前移动

代码如下：

```
class ListNode{
    int val;
    ListNode next;
    ListNode pre;
    public ListNode(int val){
        this.val = val;
    }
}

class Queue{
    ListNode tail;
    ListNode head;

    public Queue(){
        tail = null;
        head = new ListNode(0);
    }

    public void push_front(int value){
        ListNode node = new ListNode(value);
        if(tail == null){
            head.next = node;
            node.pre = head;
            tail = node;
        }else{
            node.next = head.next;
            node.next.pre = node;
            node.pre = head;
            head.next = node;
        }
    }

    public void push_back(int value){
        ListNode node = new ListNode(value);
        if(tail == null){
            head.next = node;
            node.pre = head;
            tail = node;
        }
    }
}
```

```
        }else{
            tail.next = node;
            node.pre = tail;
            tail = tail.next;
        }
    }

    public int pop_front(){
        if(head.next == null){
            return -1;
        }
        int res = head.next.val;
        head.next = head.next.next;
        if(head.next == null){
            tail = null;
            return res;
        }
        head.next.pre = head;
        return res;
    }

    public int pop_back(){
        if(tail == null){
            return -1;
        }
        int res = tail.val;
        if(tail.pre == head){
            tail = null;
            return res;
        }
        tail = tail.pre;
        return res;
    }
}
```

# Implement Queue by Linked List 492

## Question

Implement a Queue by linked list. Support the following basic methods:

`enqueue(item)`. Put a new item in the queue.

`dequeue()`. Move the first item out of the queue, return it.

Example:

`enqueue(1)`

`enqueue(2)`

`enqueue(3)`

`dequeue()` # return 1

`enqueue(4)`

`dequeue()` # return 2

## Solution

水题。同Implement stack 495。

代码如下：

```
class ListNode{
    int val;
    ListNode next;
    public ListNode(int val){
        this.val = val;
    }
}

class Queue{
    ListNode tail;
    ListNode head;

    public Queue(){
        tail = null;
        head = null;
    }

    public void enqueue(int value){
        if(head == null){
            head = new ListNode(value);
            tail = head;
        }else{
            tail.next = new ListNode(value);
            tail = tail.next;
        }
    }

    public int dequeue(){
        if(head == null){
            return -1;
        }
        int res = head.val;
        head = head.next;
        return res;
    }
}
```



# Stack Sorting 229

## Question

Sort a stack in ascending order (with biggest terms on top).

You may use at most one additional stack to hold items, but you may not copy the elements into any other data structure (e.g. array).

Example

Given stack =

| |

|3|

|1|

|2|

|4|

return:

| |

|4|

|3|

|2|

|1|

The data will be serialized to [4,2,1,3]. The last element is the element on the top of the stack.

## Solution

我的思路：

1) 用count记录已经排好序的元素个数，则`stack.size()-count`为未排序元素个数，每次将栈顶`stack.size()-count`个元素出栈

2) 将stack中全部出栈元素倒到helper stack中时寻找其中的最小者，将其加入回stack

3) 将helper stack中非最小值的元素倒回stack中

时间复杂度 $O(n^2)$ ，空间复杂度 $O(n)$

另一种方法：

1) 从origin stack中不断`pop()` element

2) 对于helper stack，如果`helper stack peek() < element`，则将helper stack中的元素全部转移到origin stack

3) 再将element `push()`到helper stack中

4) 不断重复上述步骤，直到origin stack isEmpty

5) 最后，所有的元素已经按照descending order排序好（smallest on top），只需将其转移到origin stack，则origin stack即为所需排序

时间复杂度和空间复杂度均与第一种相同。

代码如下：

第一种：



```
private void stackSorting(Stack<Integer> stack){
    Stack<Integer> stack2 = new Stack<Integer>();

    int count = 0;

    while(stack.size() - count > 0){
        int numberLeft = stack.size() - count;
        int min = Integer.MAX_VALUE;
        while(numberLeft > 0){
            int curt = stack.pop();
            min = Math.min(min, curt);
            stack2.push(curt);
            numberLeft--;
        }
        stack.push(min);
        while(!stack2.isEmpty()){
            int curt = stack2.pop();
            if(curt != min){
                stack.push(curt);
            }
        }
        count++;
    }
}
```

第二种：

...

```
public void stackSorting(Stack<Integer> stack) {  
    Stack<Integer> helpStack = new Stack<Integer>();  
    while (!stack.isEmpty()) {  
        int element = stack.pop();  
        while (!helpStack.isEmpty() && helpStack.peek() < element)  
            stack.push(helpStack.pop());  
        helpStack.push(element);  
    }  
    while (!helpStack.isEmpty()) {  
        stack.push(helpStack.pop());  
    }  
}
```

...

# Animal Shelter 230

## Question

An animal shelter holds only dogs and cats, and operates on a strictly "first in, first out" basis. People must adopt either the "oldest" (based on arrival time) of all animals at the shelter, or they can select whether they would prefer a dog or a cat (and will receive the oldest animal of that type). They cannot select which specific animal they would like. Create the data structures to maintain this system and implement operations such as enqueue, dequeueAny, dequeueDog and dequeueCat.

### Example

```
int CAT = 0
int DOG = 1

enqueue("james", DOG);
enqueue("tom", DOG);
enqueue("mimi", CAT);

dequeueAny(); // should return "james"
dequeueCat(); // should return "mimi"
dequeueDog(); // should return "tom"
```

### Challenge

Can you do it with single Queue?

## Solution

用两个queue分别存dog和cat，每个animal除了名字和类别外，还有一个time属性，以此来决定dequeueAny()的时候的出栈元素。

如果只用一个queue，需要用到循环队列的思想。将所有animal存在一个queue中，出队时，在找到第一个符合要求的出队元素之前，出队的元素全部从队尾再次入队，在找到第一个符合要求的元素后，再将剩下的元素依次出队之后再入队。

代码如下：

## 2 Queues

```
public class test {
    class Animal{
        String name;
        int type;
        int time;
        public Animal(String name, int type, int time){
            this.name = name;
            this.type = type;
            this.time = time;
        }
    }

    Queue<Animal> Cat = new LinkedList<Animal>();
    Queue<Animal> Dog = new LinkedList<Animal>();
    int time = 0;

    public static void main(String[] args){
        test t = new test();
        t.enqueue("james", 1);
        t.enqueue("tom", 1);
        t.enqueue("mimi", 0);
        System.out.println(t.dequeueAny()); // should return "j
ames"
        System.out.println(t.dequeueCat()); // should return "m
imi"
        System.out.println(t.dequeueDog()); // should return "t
om"
    }

    private void enqueue(String name, int type){
        if(type == 0){
            Cat.offer(new Animal(name, type, time));
        }
    }
}
```

```
        }else{
            Dog.offer(new Animal(name, type, time));
        }
        time++;
    }

    private String dequeueAny(){
        Animal catOldest = Cat.peek();
        Animal dogOldest = Dog.peek();
        if(catOldest.time < dogOldest.time){
            return Cat.poll().name;
        }else{
            return Dog.poll().name;
        }
    }

    private String dequeueCat(){
        return Cat.poll().name;
    }

    private String dequeueDog(){
        return Dog.poll().name;
    }
}
```

## 1 Queue

```
public class test {
    class Animal{
        String name;
        int type;
        public Animal(String name, int type){
            this.name = name;
            this.type = type;
        }
    }

    Queue<Animal> queue = new LinkedList<Animal>();
```

```
public static void main(String[] args){
    test t = new test();
    t.enqueue("james", 1);
    t.enqueue("tom", 1);
    t.enqueue("mimi", 0);
    System.out.println(t.dequeueAny()); // should return "j
ames"
    System.out.println(t.dequeueCat()); // should return "m
imi"
    System.out.println(t.dequeueDog()); // should return "t
om"
}

private void enqueue(String name, int type){
    queue.offer(new Animal(name, type));
}

private String dequeueAny(){
    return queue.poll().name;
}

private String dequeueCat(){
    return dequeueType(0);
}

private String dequeueDog(){
    return dequeueType(1);
}

private String dequeueType(int type){
    int shiftNumber = 0;
    while(queue.peek().type != type){
        queue.offer(queue.poll());
        shiftNumber++;
    }
    Animal target = queue.poll();
    shiftNumber = queue.size() - shiftNumber;
    while(shiftNumber != 0){
        queue.offer(queue.poll());
        shiftNumber--;
    }
}
```

```
        }  
        return target.name;  
    }  
}
```

# LFU Cache 24

## Question

LFU (Least Frequently Used) is a famous cache eviction algorithm.

For a cache with capacity  $k$ , if the cache is full and need to evict a key in it, the key with the lease frequently used will be kicked out.

Implement set and get method for LFU cache.

Example

Given capacity=3

set(2,2)

set(1,1)

get(2)

|| 2

get(1)

|| 1

get(2)

|| 2

set(3,3)

set(4,4)

get(3)

|| -1

get(2)

|| 2



get(1)

|| 1

get(4)

|| 4

## Solution

与LRU类似。用HashMap来存key及其对应node。用PriorityQueue(MinHeap)来对HashMap中的node以frequency为key进行排序（重构Comparator()）。

代码如下：

```
public class test {
    private class Node{
        int key;
        int value;
        int freq;
        public Node(int key, int value, int freq){
            this.key = key;
            this.value = value;
            this.freq = freq;
        }
    }

    HashMap<Integer, Node> map = new HashMap<Integer, Node>();
    PriorityQueue<Node> queue = new PriorityQueue<Node>(new Comparator<Node>(){
        public int compare(Node node1, Node node2){
            return node1.freq - node2.freq;
        }
    });
    int capacity = 3;

    public static void main(String[] args){
        test t = new test();
        t.set(2,2);
        t.set(1, 1);
    }
}
```

```
        System.out.println(t.get(2));
        System.out.println(t.get(1));
        System.out.println(t.get(2));
        t.set(3, 3);
        t.set(4, 4);
        System.out.println(t.get(3));
        System.out.println(t.get(2));
        System.out.println(t.get(1));
        System.out.println(t.get(4));
    }

    private int get(int key){
        if(!map.containsKey(key)){
            return -1;
        }
        Node curt = map.get(key);
        int res = curt.value;
        curt.freq++;
        return res;
    }

    private void set(int key, int value){
        if(map.containsKey(key)){
            map.get(key).value = value;
            map.get(key).freq++;
            return;
        }
        if(map.size() == capacity){
            Node curt = queue.poll();
            map.remove(curt.key);
        }
        Node node = new Node(key, value, 1);
        map.put(key, node);
        queue.offer(node);
    }
}
```

## Related Question



# LRU Cache 134

## Question

Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and set.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

set(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

## Solution

用HashMap来存key和和其对应的node，便于之后检索key是否已经存在。用双向链表，便于操作数组中间的元素移动和删除。

get()：如果key不存在，则返回-1；如果key存在，则将该key对应的node移到链表尾部。

set()：如果key存在，则将修改过value的该key对应node移到链表尾部；如果key不存在，分两种情况：1) 若chache已经达到其capacity，则删去链表第一个node（least recently used item），再加入该key的node；2) 若chache还没达到其capacity，则在链表尾部加入该key的node。

代码如下：

```
public class Solution {  
    //双向链表  
    private class Node{  
        Node next;  
        Node pre;  
        int key;  
        int value;  
        public Node(int key, int value){
```

```
        this.key = key;
        this.value = value;
        next = pre = null;
    }
}

private int capacity;
private HashMap<Integer, Node> map = new HashMap<Integer, Node>();

private Node head = new Node(-1, -1);
private Node tail = new Node(-1, -1);

// @param capacity, an integer
public Solution(int capacity) {
    // write your code here
    this.capacity = capacity;
    head.next = tail;
    tail.pre = head;
}

// @return an integer
public int get(int key) {
    // write your code here
    if(!map.containsKey(key)){
        return -1;
    }

    Node current = map.get(key);
    current.pre.next = current.next;
    current.next.pre = current.pre;

    moveToTail(current);

    return current.value;
}

// @param key, an integer
// @param value, an integer
// @return nothing
public void set(int key, int value) {
```

```
        // write your code here
        if(map.containsKey(key)){
            map.get(key).value = value;
            Node current = map.get(key);
            current.pre.next = current.next;
            current.next.pre = current.pre;

            moveToTail(current);
            return;
        }

        if(map.size() == capacity){
            map.remove(head.next.key);
            head.next = head.next.next;
            head.next.pre = head;
        }

        Node newNode = new Node(key, value);
        map.put(key, newNode);
        moveToTail(newNode);
    }

    private void moveToTail(Node node){
        node.pre = tail.pre;
        node.next = tail;
        tail.pre = node;
        node.pre.next = node;
    }
}
```

## Graph & Search

### Conclusion

DFS ( $O(2^n)$ ,  $O(n!)$ ) (思想:构建搜索树+判断可行性)

1. Find all possible solutions
2. Permutations / Subsets

BFS ( $O(m)$ ,  $O(n)$ )

1. Graph traversal (每个点只遍历一次)
2. Find shortest path in a simple graph

# Clone Graph 137

## Question

Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.

How we serialize an undirected graph:

Nodes are labeled uniquely.

We use # as a separator for each node, and , as a separator for node label and each neighbor of the node.

As an example, consider the serialized graph {0,1,2#1,2#2,2}.

The graph has a total of three nodes, and therefore contains three parts as separated by #.

First node is labeled as 0. Connect node 0 to both nodes 1 and 2.

Second node is labeled as 1. Connect node 1 to node 2.

Third node is labeled as 2. Connect node 2 to node 2 (itself), thus forming a self-cycle.

Visually, the graph looks like the following:

1

/ \

/ \

0 --- 2

/ \

\\_/



## Solution

要clone所有点和边，首先想到的是BFS。

BFS:

1) 首先用BFS遍历所有的点，同时复制。BFS用一个链表实现，待复制的点进入链表，同时用一个HashMap保存该原始点和其对应的复制点，出链表时将其neighbor加入链表，其中若neighbor已经进入过链表（用HashMap查询），则不重复进入。

2) 再复制所有的边。遍历BFS链表中每一个点，同时取出HashMap中该点对应的复制点，将该点neighbor对应的复制点全部加入对应复制点的neighbor。

也可以用DFS实现。方法与BFS类似，先遍历所有点并复制，再复制所有边（neighbor）。遍历时采用DFS搜索。

代码如下：

BFS:

```
public class Solution {
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
        if (node == null) {
            return null;
        }

        ArrayList<UndirectedGraphNode> nodes = new ArrayList<UndirectedGraphNode>();
        HashMap<UndirectedGraphNode, UndirectedGraphNode> map
            = new HashMap<UndirectedGraphNode, UndirectedGraphNode>();

        // clone nodes
        nodes.add(node);
        map.put(node, new UndirectedGraphNode(node.label));

        int start = 0;
        while (start < nodes.size()) {
            UndirectedGraphNode head = nodes.get(start++);
            for (int i = 0; i < head.neighbors.size(); i++) {
```

```

        UndirectedGraphNode neighbor = head.neighbors.get(i);
        if (!map.containsKey(neighbor)) {
            map.put(neighbor, new UndirectedGraphNode(neighbor.label));
            nodes.add(neighbor);
        }
    }

    // clone neighbors
    for (int i = 0; i < nodes.size(); i++) {
        UndirectedGraphNode newNode = map.get(nodes.get(i));
        for (int j = 0; j < nodes.get(i).neighbors.size(); j++) {
            newNode.neighbors.add(map.get(nodes.get(i).neighbors.get(j)));
        }
    }

    return map.get(node);
}

```

DFS:

```

class StackElement {
    public UndirectedGraphNode node;
    public int neighborIndex;
    public StackElement(UndirectedGraphNode node, int neighborIndex) {
        this.node = node;
        this.neighborIndex = neighborIndex;
    }
}

public class Solution {
    /**
     * @param node: A undirected graph node

```

```

    * @return: A undirected graph node
    */
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
        if (node == null) {
            return node;
        }

        // use dfs algorithm to traverse the graph and get all nodes.
        ArrayList<UndirectedGraphNode> nodes = getNodes(node);

        // copy nodes, store the old->new mapping information in a hash map
        HashMap<UndirectedGraphNode, UndirectedGraphNode> mapping = new HashMap<>();
        for (UndirectedGraphNode n : nodes) {
            mapping.put(n, new UndirectedGraphNode(n.label));
        }

        // copy neighbors(edges)
        for (UndirectedGraphNode n : nodes) {
            UndirectedGraphNode newNode = mapping.get(n);
            for (UndirectedGraphNode neighbor : n.neighbors) {
                UndirectedGraphNode newNeighbor = mapping.get(neighbor);
                newNode.neighbors.add(newNeighbor);
            }
        }

        return mapping.get(node);
    }

    private ArrayList<UndirectedGraphNode> getNodes(UndirectedGraphNode node) {
        Stack<StackElement> stack = new Stack<StackElement>();
        HashSet<UndirectedGraphNode> set = new HashSet<>();
        stack.push(new StackElement(node, -1));
        set.add(node);
    }

```

```
        while (!stack.isEmpty()) {
            StackElement current = stack.peek();
            current.neighborIndex++;
            // there is no more neighbor to traverse for the current node
            if (current.neighborIndex == current.node.neighbors.size()) {
                stack.pop();
                continue;
            }

            UndirectedGraphNode neighbor = current.node.neighbors.get(
                current.neighborIndex
            );
            // check if we visited this neighbor before
            if (set.contains(neighbor)) {
                continue;
            }

            stack.push(new StackElement(neighbor, -1));
            set.add(neighbor);
        }

        return new ArrayList<UndirectedGraphNode>(set);
    }
}
```

# Copy List with Random Pointer 105

## Question

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

Return a deep copy of the list.

Challenge

Could you solve it with  $O(1)$  space?

## Solution

问题不难。trick的地方在于在复制一个点时，其random应该指向的点还没有被创建，因此无法指向该点。

可以先用HashMap保存所有原始点和其复制点。然后逐点复制next和random关系。

也可以将每个点和其复制点相连，变成以下形式：

head->head'->node1->node1'->...

- 1) 原始点与复制点相连，复制点指向对应原始点的random
- 2) 调整复制点的random指向原始点random的下一个点（即复制的random点）
- 3) 拆分原始点和复制点的链表

代码如下：

HashMap:

```

public class Solution {
    /**
     * @param head: The head of linked list with a random pointer.
     * @return: A new head of a deep copy of the list.
     */

    public RandomListNode copyRandomList(RandomListNode head) {
        if(head == null){
            return head;
        }

        HashMap<RandomListNode, RandomListNode> myMap = new HashMap<RandomListNode, RandomListNode>();

        RandomListNode dummy = new RandomListNode(0);
        dummy.next = head;

        while(head != null){
            myMap.put(head, new RandomListNode(head.label));
            head = head.next;
        }

        head = dummy.next;
        RandomListNode newHead = new RandomListNode(0);
        newHead.next = myMap.get(head);
        while(head != null){
            myMap.get(head).next = myMap.get(head.next);
            myMap.get(head).random = myMap.get(head.random);
            head = head.next;
        }

        return newHead.next;
    }
}

```

Duplicate:

```

public class Solution {

```

```

/**
 * @param head: The head of linked list with a random pointer.
 * @return: A new head of a deep copy of the list.
 */

public RandomListNode copyRandomList(RandomListNode head) {
    //先构建node1->node1'->node2->node2'->...，再拆分
    private void copyNext(RandomListNode head){
        while(head != null){
            RandomListNode newNode = new RandomListNode(head.label);

            //先把复制点的random指向head的random
            newNode.random = head.random;
            newNode.next = head.next;
            head.next = newNode;
            head = head.next.next;
        }
    }

    private void copyRandom(RandomListNode head){
        while(head != null){
            if(head.next.random != null){
                //将复制点的random指向head的random之后一个点（为head的random的复制点）
                head.next.random = head.random.next;
            }
            head = head.next.next;
        }
    }

    //将list拆为原list和复制的list
    private RandomListNode split(RandomListNode head){
        RandomListNode newHead = new RandomListNode(0);
        newHead.next = head.next;
        while(head != null){
            RandomListNode temp = head.next;
            head.next = temp.next;
            if(temp.next != null){
                temp.next = temp.next.next;
            }
        }
    }
}

```

```
        head = head.next;
    }
    return newHead.next;
}

public RandomListNode copyRandomList(RandomListNode head) {
    // write your code here
    if(head == null){
        return head;
    }

    copyNext(head);
    copyRandom(head);

    return split(head);
}
}
```



# Topological Sorting 127

## Question

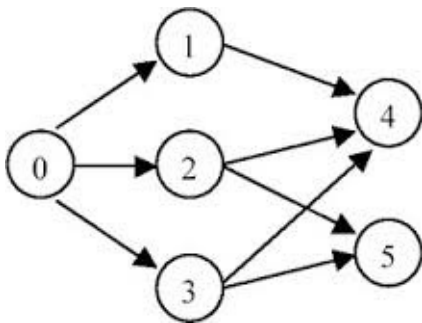
Given an directed graph, a topological order of the graph nodes is defined as follow:

For each directed edge  $A \rightarrow B$  in graph, A must before B in the order list.

The first node in the order can be any node in the graph with no nodes direct to it. Find any topological order for the given graph.

Example

For graph as follow:



The topological order can be:

[0, 1, 2, 3, 4, 5] [0, 2, 3, 1, 5, 4] ...

Challenge

Can you do it in both BFS and DFS?

## Solution

- 1) 计算所有点的入度，用HashMap保存。
- 2) 将入度为0的点加入queue中和result中，将queue中节点出队，将出队节点所有neighbor的入度减少1。
- 3) 重复2直到所有点都被加入result中。

需要注意的是，如果有对HashMap做删除操作，不能再用原来的迭代器继续迭代，需要从头用新的迭代器进行迭代。

代码如下：

```
/**
 * Definition for Directed graph.
 * class DirectedGraphNode {
 *     int label;
 *     ArrayList<DirectedGraphNode> neighbors;
 *     DirectedGraphNode(int x) { label = x; neighbors = new Arr
ArrayList<DirectedGraphNode>(); }
 * };
 */
public class Solution {
    /**
     * @param graph: A list of Directed graph node
     * @return: Any topological order for the given graph.
     */
    public ArrayList<DirectedGraphNode> topSort(ArrayList<Direct
edGraphNode> graph) {
        // write your code here
        HashMap<DirectedGraphNode, Integer> map = new HashMap<Di
rectedGraphNode, Integer>();

        //计算并保存每个点的入度
        for(int i = 0; i < graph.size(); i++){
            DirectedGraphNode curt = graph.get(i);
            // if(!map.containsKey(curt)){
            //     map.put(curt, 0);
            // }
            ArrayList<DirectedGraphNode> neighborList = curt.nei
ghbors;
            for(int j = 0; j < neighborList.size(); j++){
                DirectedGraphNode neighbor = neighborList.get(j)
;

                if(!map.containsKey(neighbor)){
                    map.put(neighbor, 1);
                }else{
                    map.put(neighbor, map.get(neighbor) + 1);
                }
            }
        }
    }
}
```

```
        }
    }
}

    ArrayList<DirectedGraphNode> result = new ArrayList<DirectedGraphNode>();
    Queue<DirectedGraphNode> queue = new LinkedList<DirectedGraphNode>();
    for(DirectedGraphNode n : graph){
        if(!map.containsKey(n)){
            queue.offer(n);
            result.add(n);
        }
    }
    //取入度为0的节点加入queue和result中
    while(!queue.isEmpty()){
        DirectedGraphNode curt = queue.poll();
        for(DirectedGraphNode neighbor : curt.neighbors){
            map.put(neighbor, map.get(neighbor) - 1);
            if(map.get(neighbor) == 0){
                queue.offer(neighbor);
                result.add(neighbor);
            }
        }
    }
    return result;
}
}
```

# Permutations 15

## Question

Given a list of numbers, return all possible permutations.

Example

For nums = [1,2,3], the permutations are:

[ [1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1] ]

Challenge

Do it without recursion.

## Solution

递归求解，[]->[1]->[1,2]->[1,2,3]->[1,3]->[1,3,2]->[2]->[2,1]->[2,1,3]->[2,3]->[2,3,1]->[3]->[3,1]->[3,1,2]->[3,2]->[3,2,1]。原理为DFS。

非递归求解。

代码如下：

Recursion:

```
class Solution {
    /**
     * @param nums: A list of integers.
     * @return: A list of permutations.
     */
    //recursion vision
    public ArrayList<ArrayList<Integer>> permute(ArrayList<Integer> nums) {
        // write your code here
        ArrayList<ArrayList<Integer>> result=new ArrayList<ArrayList<Integer>>();
        if(nums==null || nums.size()==0){
```

```

        return result;
    }

    ArrayList<Integer> list=new ArrayList<Integer>();
    helper(result,list,nums);
    return result;
}

    public void helper(ArrayList<ArrayList<Integer>> result, ArrayList<Integer> list, ArrayList<Integer> nums){
        //将当前答案list加入result时不能将原有list加入，要新建一个并copy list的值后加入
        if(list.size()==nums.size()){
            result.add(new ArrayList<Integer>(list));
            return;
        }

        for(int i=0; i<nums.size(); i++){
            //若包含当前值则continue，直到找到不包含的加入list
            if(list.contains(nums.get(i))){
                continue;
            }
            list.add(nums.get(i));
            helper(result, list, nums);
            list.remove(list.size()-1);
        }
    }
}

```

Non-Recursion:

```

class Solution {
    /**
     * @param nums: A list of integers.
     * @return: A list of permutations.
     */
    //recursion vision
    public ArrayList<ArrayList<Integer>> permute(ArrayList<Integer> nums) {

```

```
// write your code here
ArrayList<ArrayList<Integer>> result=new ArrayList<ArrayList<Integer>>();
if(nums==null || nums.size()==0){
    return result;
}

int n=nums.size();

//记录index
ArrayList<Integer> pos=new ArrayList<Integer>();
pos.add(-1);

while(!pos.isEmpty()){
    int last=pos.get(pos.size()-1);
    pos.remove(pos.size()-1);

    int next=-1;

    //看删去的元素之后是否还有可以加入的元素
    for(int i=last+1; i<n; i++){
        if(!pos.contains(i)){
            next=i;
            break;
        }
    }
    }//for

    //若没有可以加入元素，则继续删去上一位
    if(next== -1){
        continue;
    }

    //若有可以加入元素，则加入该元素，并加入其它为加入过的元素
    pos.add(next);
    for(int i=0; i<n; i++){
        if(!pos.contains(i)){
            pos.add(i);
        }
    }
    }//for
```

```
        ArrayList<Integer> list=new ArrayList<Integer>();
        for(int i=0; i<n; i++){
            list.add(nums.get(pos.get(i)));
        }
        result.add(list);
    }//while

    return result;
}
}
```

# Permutations 16

## Question

Given a list of numbers with duplicate number in it. Find all unique permutations.

Example

For numbers [1,2,2] the unique permutations are:

[ [1,2,2],

[2,1,2],

[2,2,1] ]

Challenge

Using recursion to do it is acceptable. If you can do it without recursion, that would be great!

## Solution

Recursion: 与Permutations 15类似，但是因为有重复元素，为了避免重复答案，有几点要注意。

1) 将所有重复元素排在一起，因此需要对输入数组先进行排序。

2) 用一个visit数组记录元素的状态，在list中为1，否则为0

3) 避免重复的方法：重复元素不能越过第一个元素取后面的重复元素，即若第一个重复元素不在list中，后面的重复元素不能加入list。因此在加入元素时，需要判断该元素值是否和前一个元素相等，同时前一个元素是否在list中。

Non-Recursion: 之后补上

代码如下：

```
class Solution {
```



```

/**
 * @param nums: A list of integers.
 * @return: A list of unique permutations.
 */
public ArrayList<ArrayList<Integer>> permuteUnique(ArrayList
<Integer> nums) {
    // write your code here
    ArrayList<ArrayList<Integer>> result = new ArrayList<Arr
ayList<Integer>>();
    ArrayList<Integer> list = new ArrayList<Integer>();

    if(nums == null || nums.size() == 0){
        return result;
    }
    //用一个visit数组来记录元素是否已经被加入list
    int[] visit = new int[nums.size()];
    //必须将相同元素放在一起，这样下面的判断重复的条件才能实现
    Collections.sort(nums);
    permuteHelper(result, list, nums, visit);
    return result;
}

public void permuteHelper(ArrayList<ArrayList<Integer>> resu
lt, ArrayList<Integer> list, ArrayList<Integer> nums, int[] visi
t){
    if(list.size()==nums.size()){
        result.add(new ArrayList<Integer>(list));
        return;
    }

    for(int i=0; i<nums.size(); i++){
        //若该元素未被加入list，同时前面相等值的元素也未被加入list，
        则该元素不能先被加入，否则会出现重复情况
        if(visit[i] == 1 || (i != 0 && nums.get(i) == nums.g
et(i-1) && visit[i - 1] == 0)){
            continue;
        }
        visit[i] = 1;
        list.add(nums.get(i));
        permuteHelper(result, list, nums, visit);
    }
}

```

```
        list.remove(list.size()-1);  
        visit[i] = 0;  
    }  
}  
}
```

Non-Recursion:

之后补上

## Related Question

[String Permutation II 10](#)

## Subsets 17

### Question

Given a set of distinct integers, return all possible subsets.

Example

If  $S = [1,2,3]$ , a solution is:

[ [3],

[1],

[2],

[1,2,3],

[1,3],

[2,3],

[1,2],

[] ]

Challenge

Can you do it in both recursively and iteratively?

### Solution

1. Recursion类似Permutations，但是多传入一个参数pos用于记录位置，能加入的元素只能在pos和其之后的元素中寻找。同时，subsets长度可以任意，因此list在加入result时不用像permutation那样判断是否长度等于原数组长度。
2. Non-Recursion可以用位运算来实现。输入数组长度为n，则共有 $2^n$ 个subset。从0到 $n-1$ 位上，若用1表示取该位数，0表示不取该位数，则可以将这 $2^n$ 种subset表示成 $0-2^n-1$ 之间的二进制数。以[1,2,3]为例：

```
0 -> 000 -> []  
  
1 -> 001 -> [1]  
  
2 -> 010 -> [2]  
  
..  
  
7 -> 111 -> [1, 2, 3]
```

每次取一种subset的排列，看其在0-n-1位上的哪一位不为0，就将那一位上的数加入list。看第j位是否为0，可以和 $1 \ll j$  ( $j = 0, 1, \dots, n-1$ ) 取&运算，若结果不为0，则第j位不为0。

代码如下：

Recursion:

```
class Solution {
    /**
     * @param S: A set of numbers.
     * @return: A list of lists. All valid subsets.
     */
    public ArrayList<ArrayList<Integer>> subsets(int[] nums) {
        // write your code here
        ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
        ArrayList<Integer> list = new ArrayList<Integer>();

        if(nums == null || nums.length == 0){
            return result;
        }
        Arrays.sort(nums);
        helper(result, list, nums, 0);
        return result;
    }

    public void helper(ArrayList<ArrayList<Integer>> result, ArrayList<Integer> list, int[] nums, int pos){
        result.add(new ArrayList<Integer>(list));
        //用pos记录当前位置，所有之后加入的元素只能在pos之后的元素中寻找
        for(int i = pos; i < nums.length; i++){
            list.add(nums[i]);
            helper(result, list, nums, i+1);
            list.remove(list.size()-1);
        }
    }
}
```

Non-Recursion:

```
class Solution {
    /**
     * @param S: A set of numbers.
     * @return: A list of lists. All valid subsets.
     */
    public ArrayList<ArrayList<Integer>> subsets(int[] nums) {
        // write your code here
        ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
        if(nums == null || nums.length == 0){
            return result;
        }

        int n = nums.length;
        Arrays.sort(nums);
        for(int i = 0; i < (1 << n); i++){
            ArrayList<Integer> list = new ArrayList<Integer>();
            for(int j = 0; j < n; j++){
                if((i & (1 << j)) != 0){
                    list.add(nums[j]);
                }
            }
            result.add(list);
        }

        return result;
    }
}
```

# Subsets II 18

## Question

Given a list of numbers that may has duplicate numbers, return all possible subsets.

Example

If  $S = [1,2,2]$ , a solution is:

[ [2],

[1],

[1,2,2],

[2,2],

[1,2],

[] ]

Challenge

Can you do it in both recursively and iteratively?

## Solution

Recursion: 与Subsets 17类似，但是因为有重复元素，为了避免出现重复解，有几点需要注意。

1) 将所有重复元素排在一起，因此首先对输入数组进行排序

2) 避免重复的方法为：重复元素只能取第一个元素，因此在加入元素前要判断该元素值是否和之前元素相等。

Non-Recursion:

代码如下：

Recursion:



```

class Solution {
    /**
     * @param S: A set of numbers.
     * @return: A list of lists. All valid subsets.
     */
    public ArrayList<ArrayList<Integer>> subsetsWithDup(ArrayList<Integer> S) {
        // write your code here
        ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
        ArrayList<Integer> list = new ArrayList<Integer>();
        //当相同值元素排列在一起时下面的避免重复方法才有效，因此需要提前对S
        进行排序。
        Collections.sort(S);

        if(S == null || S.size() == 0){
            return result;
        }

        helper(result, list, S, 0);
        return result;
    }

    public void helper(ArrayList<ArrayList<Integer>> result, ArrayList<Integer> list, ArrayList<Integer> S, int pos){
        result.add(new ArrayList<Integer>(list));
        //连续相同元素在一起时，若第一个没有被加入list，则不能把后面同值元素
        先加入，否则会出现重复。
        for(int i = pos; i < S.size(); i++){
            if(i != pos && S.get(i) == S.get(i-1)){
                continue;
            }
            list.add(S.get(i));
            helper(result, list, S, i+1);
            list.remove(list.size()-1);
        }
    }
}

```



# N-Queens 33

## Question

The n-queens puzzle is the problem of placing n queens on an  $n \times n$  chessboard such that no two queens attack each other.

Given an integer n, return all distinct solutions to the n-queens puzzle.

Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space respectively.

Example

There exist two distinct solutions to the 4-queens puzzle:

```
[ // Solution 1
```

```
[".Q..",
```

```
"...Q",
```

```
"Q...",
```

```
"..Q." ],
```

```
// Solution 2
```

```
["..Q.",
```

```
"Q...",
```

```
"...Q",
```

```
".Q.." ] ]
```

Challenge

Can you do it without recursion?

## Solution

Recursion: 确定上一行Queen的位置，递归寻找下一行Queen的位置。寻找某一行Queen位置时，必须保证该行Queen占据的位置不和该行之前所有行的Queen有冲突，即不能在同一列，也不能在对角线。

Non-Recursion：之后补上

代码如下：

Recursion:

```
class Solution {
    /**
     * Get all distinct N-Queen solutions
     * @param n: The number of queens
     * @return: All distinct solutions
     * For example, A string '...Q' shows a queen on forth posit
ion
     */
    ArrayList<ArrayList<String>> solveNQueens(int n) {
        // write your code here
        ArrayList<ArrayList<String>> resultList = new ArrayList<
>();
        if (n <= 0) {
            return resultList;
        }
        int[] row = new int[n];
        //从第0行开始寻找Queen的位置
        solveNQueensCore(resultList, row, n, 0);
        return resultList;
    }

    private void solveNQueensCore(ArrayList<ArrayList<String>> r
esultList,
                                int[] row,
                                int n,
                                int index) {
        //前n行寻找完成，即找到一种solution
        if (index == n) {
            ArrayList<String> singleResult = translateString(row
);
```

```

        resultList.add(singleResult);
        return;
    }
    //逐列寻找第index行Queen可能的位置
    for (int i = 0; i < n; i++) {
        if (isValid(row, index, i)) {
            //记录该行Queen可能的位置
            row[index] = i;
            //递归寻找下一行Queen可能的位置
            solveNQueensCore(resultList, row, n, index + 1);
            //将该位置复原，继续寻找下一个可能的答案
            row[index] = 0;
        }
    }
}
//将一种solution用string表示出来
private ArrayList<String> translateString(int[] row) {
    ArrayList<String> resultList = new ArrayList<>();
    for (int i = 0; i < row.length; i++) {
        StringBuilder sb = new StringBuilder();
        for (int j = 0; j < row.length; j++) {
            if (j == row[i]) {
                sb.append('Q');
            }
            else {
                sb.append('.');
            }
        }
        resultList.add(sb.toString());
    }
    return resultList;
}
//判断Queen在第rowNum行第columnNum列是否可行
private boolean isValid(int[] row, int rowNum, int columnNum
) {
    //需要保证和rowNum行之前所有行Queen不冲突
    for (int i = 0; i < rowNum; i++) {
        //判断列是否冲突
        if (row[i] == columnNum) {
            return false;
        }
    }
    return true;
}

```

```
        }  
        //判断对角线是否冲突  
        if (Math.abs(row[i] - columnNum) == Math.abs(i - row  
Num)) {  
            return false;  
        }  
    }  
    return true;  
}  
  
}
```

Non-Recursion:

之后补上

# N-Queens II 34

## Question

Follow up for N-Queens problem.

Now, instead outputting board configurations, return the total number of distinct solutions.

Example

For  $n=4$ , there are 2 distinct solutions.

## Solution

与N-Queens 33相同，不用具体记录每种解法，只要记录次数。

代码如下：

```
class Solution {  
    /**  
     * Calculate the total number of distinct N-Queen solutions.  
     * @param n: The number of queens.  
     * @return: The total number of distinct solutions.  
     */  
    int sum;  
    public int totalNQueens(int n) {  
        //write your code here  
        sum = 0;  
        int[] row = new int[n];  
        NQueenHelper(row, 0);  
  
        return sum;  
    }  
  
    private void NQueenHelper(int[] row, int rowNum){  
        int n = row.length;
```

```
        if(rowNum == n){
            sum++;
            return;
        }

        for(int i = 0; i < n; i++){
            if(isValid(row, rowNum, i)){
                row[rowNum] = i;
                NQueenHelper(row, rowNum + 1);
                row[rowNum] = 0;
            }
        }
    }

    private boolean isValid(int[] row, int rowNum, int columnNum
){
        for(int i = 0; i < rowNum; i++){
            if(row[i] == columnNum){
                return false;
            }

            if(Math.abs(row[i] - columnNum) == Math.abs(i - rowN
um)){
                return false;
            }
        }
        return true;
    }
};
```



# Palindrome Partitioning 136

## Question

Given a string *s*, partition *s* such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of *s*.

Example

Given *s* = "aab", return:

```
[["aa","b"],  
["a","a","b"]]
```

## Solution

Recursion

Recursion: 类似Subsets。递归含义位：以*path*开头，*s*从第*pos*到结尾的字符串是否为回文串。

Non-Recursion：用位运算的方法，但是总时间会超时。

代码如下：

Recursion:

```
public class Solution {  
    /**  
     * @param s: A string  
     * @return: A list of lists of string  
     */  
    public ArrayList<ArrayList<String>> partition(String s) {  
        // write your code here  
        ArrayList<ArrayList<String>> result = new ArrayList<Arra  
yList<String>>();
```

```

        if (s == null) {
            return result;
        }

        ArrayList<String> path = new ArrayList<String>();
        helper(s, path, 0, result);

        return result;
    }

    private void helper(String s, ArrayList<String> path, int pos, ArrayList<ArrayList<String>> result){
        if(pos == s.length()){
            result.add(new ArrayList<String>(path));
            return;
        }

        for(int i = pos + 1; i <= s.length(); i++){
            String sb = s.substring(pos, i);
            if(!isPalindrome(sb)){
                continue;
            }

            path.add(sb);
            helper(s, path, i, result);
            path.remove(path.size() - 1);
        }
    }

    private boolean isPalindrome(String s){
        int length = s.length();
        if(length == 1){
            return true;
        }

        for(int i = 0, j = length - 1; i <= j; i++, j--){
            if(s.charAt(i) != s.charAt(j)){
                return false;
            }
        }
    }

```

```

        return true;
    }
}

```

Non-Recursion: 超时

```

public class Solution {
    /**
     * @param s: A string
     * @return: A list of lists of string
     */
    public ArrayList<ArrayList<String>> partition(String s) {
        // write your code here
        ArrayList<ArrayList<String>> result = new ArrayList<ArrayList<String>>();
        if(s == null || s.length() == 0){
            return result;
        }

        int n = s.length() - 1;
        for(int i = 0; i < Math.pow(2, n); i++){
            int last = 0;
            ArrayList<String> list = new ArrayList<String>();
            int j = 0;
            for(; j < n; j++){
                if((i & (1 <= j)) != 0){
                    String sb = s.substring(last, j + 1);
                    if(!isPalindrome(sb)){
                        break;
                    }
                    list.add(sb);
                    last = j + 1;
                }
            }
            if(j == n){
                if(isPalindrome(s.substring(last))){
                    list.add(s.substring(last));
                    result.add(list);
                }else{

```

```

        continue;
    }
    }else{
        continue;
    }
}

return result;
}

private boolean isPalindromeList(ArrayList<String> list){
    for(String s : list){
        if(!isPalindrome(s)){
            return false;
        }
    }
    return true;
}

private boolean isPalindrome(String s){
    int length = s.length();
    if(length == 1){
        return true;
    }

    for(int i = 0, j = length - 1; i <= j; i++, j--){
        if(s.charAt(i) != s.charAt(j)){
            return false;
        }
    }
    return true;
}
}

```

## String Permutation II 10

### Question:

Given a string, find all permutations of it without duplicates.

Example

Given "abb", return ["abb", "bab", "bba"].

Given "aabb", return ["aabb", "abab", "baba", "bbaa", "abba", "baab"].

### Solution:

与Permutation II 16解法一样。

代码如下：

```
public class Solution {
    /**
     * @param str a string
     * @return all permutations
     */

    public List<String> stringPermutation2(String str) {
        // Write your code here
        char[] string = str.toCharArray();
        boolean[] isUsed = new boolean[string.length];
        Arrays.sort(string);
        List<String> result = new ArrayList<String>();
        String temp = new String();
        stringPermutation2Helper(result, temp, string, isUsed);
        return result;
    }

    private void stringPermutation2Helper(List<String> result,
                                           String temp,
                                           char[] string,
                                           boolean[] isUsed) {
        if (temp.length() == string.length) {
            result.add(temp);
            return;
        }
        for (int i = 0; i < string.length; i++) {
            if (isUsed[i] == true || i != 0 &&
                isUsed[i - 1] == false &&
                string[i] == string[i - 1]) {
                continue;
            }
            isUsed[i] = true;
            stringPermutation2Helper(result, temp + string[i], s
tring, isUsed);
            isUsed[i] = false;
        }
    }
}
```

Revised Version:

```
public class Solution {

    /**
     * @param str a string
     * @return all permutations
     */

    public List<String> stringPermutation2(String str) {
        // Write your code here
        List<String> result = new ArrayList<String>();
        char[] s = str.toCharArray();
        Arrays.sort(s);
        result.add(String.valueOf(s));
        while ((s = nextPermutation(s)) != null) {
            result.add(String.valueOf(s));
        }
        return result;
    }

    public char[] nextPermutation(char[] nums) {
        int index = -1;
        for(int i = nums.length - 1; i > 0; i--){
            if(nums[i] > nums[i-1]){
                index = i-1;
                break;
            }
        }
        if(index == -1){
            return null;
        }
        for(int i = nums.length - 1; i > index; i--){
            if(nums[i] > nums[index]){
                char temp = nums[i];
                nums[i] = nums[index];
                nums[index] = temp;
                break;
            }
        }
    }
}
```

```
        reverse(nums, index+1, nums.length-1);
        return nums;
    }

    public void reverse(char[] num, int start, int end) {
        for (int i = start, j = end; i < j; i++, j--) {
            char temp = num[i];
            num[i] = num[j];
            num[j] = temp;
        }
    }
}
```



## k Sum II 90

### Question

Given  $n$  unique integers, number  $k$  ( $1 \leq k \leq n$ ) and target.

Find all possible  $k$  integers where their sum is target.

Example

Given  $[1, 2, 3, 4]$ ,  $k = 2$ , target = 5. Return:

$[[1, 4],$

$[2, 3]]$

### Solution

DFS递归求解。

代码如下：

```
public class Solution {  
    /**  
     * @param A: an integer array.  
     * @param k: a positive integer (k <= length(A))  
     * @param target: a integer  
     * @return a list of lists of integer  
     */  
    public ArrayList<ArrayList<Integer>> kSumII(int[] A, int k,  
int target) {  
        // write your code here  
        ArrayList<ArrayList<Integer>> result = new ArrayList<Arr  
ayList<Integer>>();  
        if(A == null || A.length == 0){  
            return result;  
        }  
    }  
}
```

```
        ArrayList<Integer> list = new ArrayList<Integer>();
        Arrays.sort(A);
        helper(result, list, A, k, target, 0);

        return result;
    }

    private void helper(ArrayList<ArrayList<Integer>> result, ArrayList<Integer> list, int[] A, int k, int target, int pos){

        if(k == 0 && target == 0){
            result.add(new ArrayList<Integer>(list));
            return;
        }

        if(k > 0 && target > 0){
            for(int i = pos; i < A.length; i++){
                if(A[i] > target){
                    return;
                }
                list.add(A[i]);
                helper(result, list, A, k - 1, target - A[i], i
+ 1);

                list.remove(list.size() - 1);
            }
        }
    }
}
```

## Two Pointers

## Two Sum II 443

### Question

Given an array of integers, find how many pairs in the array such that their sum is bigger than a specific target number. Please return the number of pairs.

Example

Given numbers = [2, 7, 11, 15], target = 24. Return 1. (11 + 15 is the only pair)

Challenge

Do it in  $O(1)$  extra space and  $O(n \log n)$  time.

### Solution

原理和two sum一样，用前后指针。

1. 若前指针和后指针之和小于target，则增大前指针，
2. 若前指针和后指针之和大于target，则前指针和前后指针之间的所有数与后指针之和都大于target，将这段pair数加入sum，再减小后指针
3. 重复1—2直到两个指针相遇

代码如下：

```
public class Solution {  
    /**  
     * @param nums: an array of integer  
     * @param target: an integer  
     * @return: an integer  
     */  
    public int twoSum2(int[] nums, int target) {  
        // Write your code here  
        if(nums == null || nums.length == 0){  
            return 0;  
        }  
  
        Arrays.sort(nums);  
  
        int sum = 0;  
        int i = 0;  
        int j = nums.length - 1;  
        while(i < j){  
            if(nums[i] + nums[j] <= target){  
                i++;  
            }else{  
                sum += j - i;  
                j--;  
            }  
        }  
  
        return sum;  
    }  
}
```

# Remove Element 172

## Question

Given an array and a value, remove all occurrences of that value in place and return the new length.

The order of elements can be changed, and the elements after the new length don't matter.

Example

Given an array [0,4,4,0,0,2,4,4], value=4

return 4 and front four elements of the array is [0,0,0,2]

## Solution

这题本身很简单，用前后两个指针就可以解决。前指针从前往后找第一个值等于 `elem` 的元素，后指针从后往前找第一个值不等于 `elem` 的元素，交换两个元素的值，直到前指针位置超过后指针为止。

但是有一点需要注意，那就是最后返回的长度需要讨论：

1. 如果在 `start==end` 的地方和 `val` 相等，则返回 `start`
2. 如果在 `start==end` 的地方和 `val` 不等，则返回 `start+1`

代码如下：

```
public class Solution {  
    public int removeElement(int[] nums, int val) {  
        if(nums == null || nums.length == 0){  
            return 0;  
        }  
  
        int start = 0;  
        int end = nums.length - 1;  
        while(start < end){  
            while(start < end && nums[start] != val){  
                start++;  
            }  
  
            while(start < end && nums[end] == val){  
                end--;  
            }  
  
            int temp = nums[start];  
            nums[start] = nums[end];  
            nums[end] = temp;  
        }  
        return nums[start] == val? start : start + 1;  
    }  
}
```

# Triangle Count 382

## Question

Given an array of integers, how many three numbers can be found in the array, so that we can build an triangle whose three edges length is the three numbers that we find?

Example

Given array  $S = [3, 4, 6, 7]$ , return 3. They are:

$[3, 4, 6]$

$[3, 6, 7]$

$[4, 6, 7]$

Given array  $S = [4, 4, 4, 4]$ , return 4. They are:

$[4(1), 4(2), 4(3)]$

$[4(1), 4(2), 4(4)]$

$[4(1), 4(3), 4(4)]$

$[4(2), 4(3), 4(4)]$

## Solution

idea：想法与two sumII一样。target设为当前数组最大值，寻找之前元素中两个元素之和大于target的对数，然后再将target元素左移，寻找下一个target。

1. 对数组从小到大排序
2. 将元素最大值设为target，寻找之前元素中两个元素之和大于target的对数并计入总数
3. 将target元素左移一位，重复2直到target位置为2



#### 4. 返回总数即为结果

代码如下：

```
public class Solution {  
    /**  
     * @param S: A list of integers  
     * @return: An integer  
     */  
    public int triangleCount(int S[]) {  
        // write your code here  
        if(S == null || S.length <= 2){  
            return 0;  
        }  
  
        Arrays.sort(S);  
  
        int sum = 0;  
        for(int i = S.length - 1; i >= 2; i--){  
            int target = S[i];  
            int left = 0;  
            int right = i - 1;  
            while(left < right){  
                if(S[left] + S[right] > target){  
                    sum += right - left;  
                    right--;  
                }else{  
                    left++;  
                }  
            }  
        }  
  
        return sum;  
    }  
}
```

# Valid Palindrome 415

## Question

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

Notice

Have you consider that the string might be empty? This is a good question to ask during an interview.

For the purpose of this problem, we define empty string as valid palindrome.

Example

"A man, a plan, a canal: Panama" is a palindrome.

"race a car" is not a palindrome.

Challenge

$O(n)$  time without extra memory.

## Solution

前后指针，用Character.isLetterOrDigit来判断是否是字符和数字。

代码如下：

```
public class Solution {  
    /**  
     * @param s A string  
     * @return Whether the string is a valid palindrome  
     */  
    public boolean isPalindrome(String s) {  
        // Write your code here  
        if(s == null){  
            return false;  
        }  
  
        if(s.length() == 0){  
            return true;  
        }  
  
        s = s.toLowerCase();  
  
        for(int i = 0, j = s.length() - 1; i < j; i++, j--){  
            while(i < j && !Character.isLetterOrDigit(s.charAt(i  
))) {  
                i++;  
            }  
  
            while(i < j && !Character.isLetterOrDigit(s.charAt(j  
))) {  
                j--;  
            }  
  
            if(s.charAt(i) != s.charAt(j)){  
                return false;  
            }  
        }  
  
        return true;  
    }  
}
```



# Linked List Cycle II 103

## Question

Given a linked list, return the node where the cycle begins.

If there is no cycle, return null.

Example

Given -21->10->4->5, tail connects to node index 1 , return 10

Challenge

Follow up:

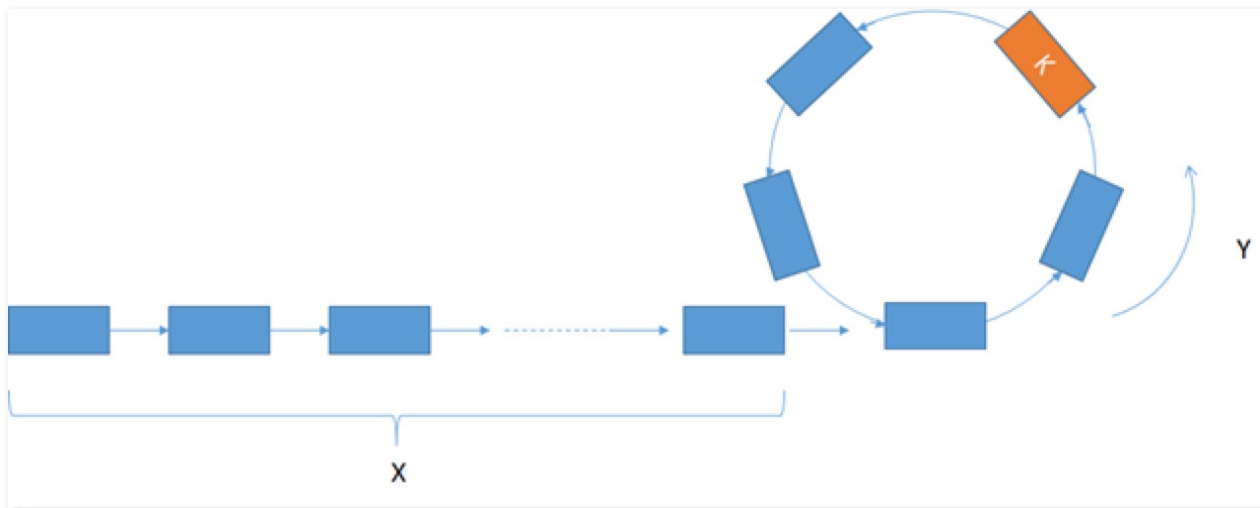
Can you solve it without using extra space?

## Solution

1. 先用快慢指针判断是不是存在环
2. 当slow和fast第一次相遇时，将slow放回Start处，fast向下移动一步，两个指针再一起移动，直到再次相遇，就是交点

为什么slow放回start处，fast要向下移动一位：slow和fast第一次相当于从head之前的一个dummy node一起出发，slow移动一步到head，fast移动两步到head.next，所以第一次相遇之后，相当于要将slow放回dummy，然后slow和fast一起移动（一步之后slow到达head，fast到达fast.next）。

证明：



现在有两个指针，第一个指针，每走一次走一步，第二个指针每走一次走两步，如果他们走了 $t$ 次之后相遇在 $K$ 点

那么指针一走的路是  $t = X + nY + K$  ①

指针二走的路是  $2t = X + mY + K$  ②  $m, n$  为未知数

把等式一代入到等式二中, 有

$$2X + 2nY + 2K = X + mY + K$$

$$X + (2n - m)Y + K = 0;$$

$$X + K = (m - 2n)Y$$

这就清晰了， $X$ 和 $K$ 的关系是基于 $Y$ 互补的。等于说，两个指针相遇以后，再往下走 $X$ 步就回到Cycle的起点了。

代码如下：

```
/**
 * Definition for ListNode.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int val) {
 *         this.val = val;
 *         this.next = null;
 *     }
 * }
 */
```

```
public class Solution {  
    /**  
     * @param head: The first node of linked list.  
     * @return: The node where the cycle begins.  
     *         if there is no cycle, return null  
     */  
    public ListNode detectCycle(ListNode head) {  
        // write your code here  
        if(head == null || head.next == null){  
            return null;  
        }  
  
        ListNode slow = head;  
        ListNode fast = head.next;  
  
        while(slow != fast){  
            if(fast != null && fast.next != null){  
                slow = slow.next;  
                fast = fast.next.next;  
            }else{  
                return null;  
            }  
        }  
  
        slow = head;  
        while(slow != fast){  
            slow = slow.next;  
            fast = fast.next;  
        }  
  
        return slow;  
    }  
}
```

# Longest Substring with At Most K Distinct Characters 386

## Question

Given a string  $s$ , find the length of the longest substring  $T$  that contains at most  $k$  distinct characters.

Example

For example, Given  $s = \text{"eceba"}$ ,  $k = 3$ ,

$T$  is  $\text{"eceb"}$  which its length is 4.

Challenge

$O(n)$ ,  $n$  is the size of the string  $s$ .

## Solution

用双指针解题。用一个hashmap记录遇到的字符及其出现次数。

1. 指针 $left$ 和 $right$ 都从0开始， $right$ 向后遍历数组。
2. 当 $right$ 遇到字符是之前出现过的，则直接在hashmap中将其数量加1；若 $right$ 遇到字符是之前没有出现过的，则分情况讨论1) 若目前遇到过字符种类不足 $k$ 个，则直接加入；2) 若目前遇到过的字符种类大于 $k$ 个，则移动左边界减少字符，直到字符种类少于 $k$ 个后再加入。
3. 每次改变左边界时更新 $max$ 的值。但是最后一次当 $right$ 到数组尾部时，这一次不会改变左边界，所以要在循环结束后最后一次更新 $max$ 。

代码如下：

```
public class Solution {  
    /**  
     * @param s : A string  
     * @return : The length of the longest substring
```



```

        *           that contains at most k distinct characters.
        */
    public int lengthOfLongestSubstringKDistinct(String s, int k
) {
        // write your code here
        if(s == null || s.length() == 0 || k == 0){
            return 0;
        }

        if(s.length() <= k){
            return s.length();
        }

        HashMap<Character, Integer> map = new HashMap<Character,
Integer>();
        int left = 0;
        int right = 0;
        int max = 0;
        while(right < s.length()){
            //如果遇到的字符是之前出现过的，则直接将其数量加1
            if(map.containsKey(s.charAt(right))){
                map.put(s.charAt(right), map.get(s.charAt(right)
) + 1);
            }else{
                //如果遇到的新字符之前没有出现过则分情况讨论
                //若目前遇到过字符种类不足k个，则直接加入
                if(map.size() < k){
                    map.put(s.charAt(right), 1);
                }else{
                    //若目前遇到过的字符种类大于k个，则移动左边界减少字符，直到字
符种类少于k个后再加入
                    max = Math.max(max, right - left);
                    while(map.size() >= k){
                        map.put(s.charAt(left), map.get(s.charAt
(left)) - 1);

                        if(map.get(s.charAt(left)) == 0){
                            map.remove(s.charAt(left));
                        }
                        left++;
                    }
                }
            }
        }
    }

```

```
        map.put(s.charAt(right), 1);
    }
}
right++;
}

//right到底之后要统计最后一次substring的长度
max = Math.max(max, right - left);

return max;
}
}
```

# Subarray Sum II 404

## Question

Given an integer array, find a subarray where the sum of numbers is in a given interval. Your code should return the number of possible answers. (The element in the array should be positive)

Example

Given [1,2,3,4] and interval = [1,3], return 4. The possible answers are:

[0, 0] [0, 1] [1, 1] [2, 2]

## Solution

方法1: 和I一样，用prefix来解。首先计算prefix，然后根据prefix来计算每一段subarray的sum。右边prefix - 左边prefix等于中间这一段subarray的sum。左边prefix从0枚举到n-1，右边prefix从当前左边往右一位开始枚举到n，这样可以得到所有subarray的sum，每一个sum如果在interval之内则count增加1个。O(n^2)。

方法2: 和1思想其实一样，每次右边界j向右移动一位，然后枚举左边界从0到j-1，计算左右边界array的sum，看这些sum是否在interval中。

代码如下：

```
public class Solution {  
    /**  
     * @param A an integer array  
     * @param start an integer  
     * @param end an integer  
     * @return the number of possible answer  
     */  
    public int subarraySumII(int[] A, int start, int end) {  
        // Write your code here  
        if(A == null || A.length == 0 || start > end){  
            return 0;  
        }  
    }  
}
```

```
}
//preix version
int[] sum = new int[A.length + 1];
sum[0] = 0;
for(int i = 1; i <= A.length; i++){
    sum[i] = sum[i - 1] + A[i - 1];
}

int count = 0;
for(int i = 0; i < A.length; i++){
    for(int j = i + 1; j <= A.length; j++){
        int diff = sum[j] - sum[i];
        if(diff >= start && diff <= end){
            count++;
        }
    }
}

return count;

//My version
// int j = 0;
// int count = 0;
// int sum = 0;
// while(j < A.length){
//     sum += A[j];
//     if(sum >= start && sum <= end){
//         count++;
//     }

//     int i = 0;
//     int temp = sum - A[i];
//     while(i < j){
//         if(temp >= start && temp <= end){
//             count++;
//         }
//         i++;
//         temp -= A[i];
//     }
//     j++;
```

```
        // }  
        // return count;  
    }  
}
```

## 4 Sum

### Question

Given an array  $S$  of  $n$  integers, are there elements  $a$ ,  $b$ ,  $c$ , and  $d$  in  $S$  such that  $a + b + c + d = \text{target}$ ? Find all unique quadruplets in the array which gives the sum of target.

Note: The solution set must not contain duplicate quadruplets.

For example, given array  $S = [1, 0, -1, 0, -2, 2]$ , and  $\text{target} = 0$ .

A solution set is:  $[[-1, 0, 0, 1], [-2, -1, 1, 2], [-2, 0, 0, 2]]$

### Solution

这道题和2sum，3sum一样，也是用two pointers的方法。

固定第一个数，移动第二个数（第一个数后面），在第二个数之后的数之中用2sum寻找target。有几点要注意：

1. 注意有重复的数，为了避免重复答案，只能取第一个重复的数
2. 注意conor case，数组长度小于4不行，排序后若最小值的4倍大于target或者最大值的4倍小于target不行

代码如下：

```
public class Solution {
    public List<List<Integer>> fourSum(int[] nums, int target) {
        List<List<Integer>> res = new ArrayList<List<Integer>>();
        ;
        if(nums == null || nums.length < 4){
            return res;
        }

        Arrays.sort(nums);
        if(4 * nums[0] > target || 4 * nums[nums.length - 1] < t
```

```

    target){
        return res;
    }

    for(int i = 0; i < nums.length - 3; i++){
        //skip duplicate
        if(i != 0 && nums[i] == nums[i - 1]){
            continue;
        }

        for(int j = i + 1; j < nums.length - 2; j++){
            if(j != i + 1 && nums[j] == nums[j - 1]){
                continue;
            }

            int head = j + 1;
            int tail = nums.length - 1;
            while(head < tail){
                int sum = nums[i] + nums[j] + nums[head] + n
ums[tail];
                if(sum == target){
                    List<Integer> list = new ArrayList<Integ
er>();

                    list.add(nums[i]);
                    list.add(nums[j]);
                    list.add(nums[head]);
                    list.add(nums[tail]);
                    res.add(list);
                    head++;
                    tail--;
                    while(head < tail && nums[head] == nums[
head - 1]){
                        head++;
                    }
                    while(head < tail && nums[tail] == nums[
tail + 1]){
                        tail--;
                    }
                }else if(sum < target){
                    head++;
                }
            }
        }
    }
}

```

```
        }else{
            tail--;
        }
    }
}
return res;
}
```



# Remove Nth Node From End of List 174

## Question

Given a linked list, remove the  $n$ th node from the end of list and return its head.

Notice

The minimum number of nodes in list is  $n$ .

Example

Given linked list: 1->2->3->4->5->null, and  $n = 2$ .

After removing the second node from the end, the linked list becomes 1->2->3->5->null.

Challenge

Can you do it without getting the length of the linked list?

## Solution

这道题一开始可以考虑reverse之后再删去第 $n$ 个。

如果只能走一趟，那可以用two pointer。总体思想为：要删去从后往前第 $n$ 个元素，即删去从前往后第 $\text{size}-n+1$ 个元素。一个指针先从前往后走 $n$ 步，然后此时从 $n$ 到list尾部（null）的距离为 $\text{size}-n+1$ ，这时候另一个指针从dummy（第一个元素之前的元素）和第一个指针一起往后移动直到第一个指针到达list尾部为止，此时第二个指针到达第 $\text{size}-n$ 个元素，只要删去其后面的元素即可。

代码如下：

```
public class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        if (n <= 0) {
            return null;
        }

        ListNode dummy = new ListNode(0);
        dummy.next = head;

        ListNode preDelete = dummy;
        for (int i = 0; i < n; i++) {
            if (head == null) {
                return null;
            }
            head = head.next;
        }
        while (head != null) {
            head = head.next;
            preDelete = preDelete.next;
        }
        preDelete.next = preDelete.next.next;
        return dummy.next;
    }
}
```

**Sort**

# Quick Sort

# Nuts & Bolts Problem 399

## Question

Given a set of  $n$  nuts of different sizes and  $n$  bolts of different sizes. There is a one-one mapping between nuts and bolts. Comparison of a nut to another nut or a bolt to another bolt is not allowed. It means nut can only be compared with bolt and bolt can only be compared with nut to see which one is bigger/smaller.

We will give you a compare function to compare nut with bolt.

Example Given nuts = ['ab','bc','dd','gg'], bolts = ['AB','GG', 'DD', 'BC'].

Your code should find the matching bolts and nuts.

one of the possible return:

nuts = ['ab','bc','dd','gg'], bolts = ['AB','BC','DD','GG'].

we will tell you the match compare function. If we give you another compare function.

the possible return is the following:

nuts = ['ab','bc','dd','gg'], bolts = ['BC','AA','DD','GG'].

So you must use the compare function that we give to do the sorting.

The order of the nuts or bolts does not matter.

## Solution

将nuts和bolts用quick sort排序成一一对应关系。

1. 首先以bolts的左边界元素作为基准，对nuts进行排序。在nuts中寻找该bolt对应的nut，找到之后将该nut交换到其左边界，然后以该nut为pivot进行quick sort，因为compare只能比较nut和bult，所以还是以之前的bolt作为基准，将该bolt小的nut交换到前部，将该bolt大的nut交换到后部，最后将pivot填入到相遇的位置，并返回该位置。

2. 与1类似，以1返回的位置的nut为基准，对bolt进行排序。最后将比该nut小的bolt放到前部，将比该nut大的bolt放到后部，将与该nut对应的bolt放到相遇位置。
3. 这样就有1对nut和bolt一一对应，并且在数组中该nut之前的部分都是比它小的，之后的部分都是比它大的，bolt也是一样。然后再重复1-2，递归地对前半部和后半部分进行quick sort。

代码如下：

```
/**
 * public class NBCompare {
 *     public int cmp(String a, String b);
 * }
 * You can use compare.cmp(a, b) to compare nuts "a" and bolts "
b",
 * if "a" is bigger than "b", it will return 1, else if they are
equal,
 * it will return 0, else if "a" is smaller than "b", it will re
turn -1.
 * When "a" is not a nut or "b" is not a bolt, it will return 2,
which is not valid.
 */
public class Solution {
    /**
     * @param nuts: an array of integers
     * @param bolts: an array of integers
     * @param compare: a instance of Comparator
     * @return: nothing
     */
    public void sortNutsAndBolts(String[] nuts, String[] bolts,
NBComparator compare) {
        // write your code here
        if(nuts == null || nuts.length == 0 || bolts == null ||
bolts.length == 0 || nuts.length != bolts.length){
            return;
        }

        quickSort(nuts, bolts, compare, 0, nuts.length - 1);
    }
}
```

```

    private void quickSort(String[] nuts, String[] bolts, NBComparator compare, int left, int right){

        if(left >= right){
            return;
        }

        int index = partition(nuts, bolts[left], compare, left, right);
        partition(bolts, nuts[index], compare, left, right);

        quickSort(nuts, bolts, compare, left, index - 1);
        quickSort(nuts, bolts, compare, index + 1, right);
    }

    private int partition(String[] str, String pivot, NBComparator compare, int left, int right){
        for(int i = left; i <= right; i++){
            if(compare.cmp(str[i], pivot) == 0 || compare.cmp(pivot, str[i]) == 0){
                swap(str, left, i);
                break;
            }
        }

        String curt = str[left];
        int start = left;
        int end = right;
        while(start < end){
            while(start < end && (compare.cmp(str[end], pivot) == 1 || compare.cmp(pivot, str[end]) == -1)){
                end--;
            }
            str[start] = str[end];

            while(start < end && (compare.cmp(str[start], pivot) == -1 || compare.cmp(pivot, str[start]) == 1)){
                start++;
            }
        }
    }

```

```
        str[end] = str[start];
    }

    str[start] = curt;
    return start;
}

private void swap(String[] array, int a, int b){
    String temp = array[a];
    array[a] = array[b];
    array[b] = temp;
}
};
```



# Kth Largest Element 5

## Question

Find K-th largest element in an array.

Notice

You can swap elements in the array

Example

In array [9,3,2,4,8], the 3rd largest element is 4.

In array [1,2,3,4,5], the 1st largest element is 5, 2nd largest element is 4, 3rd largest element is 3 and etc.

Challenge

$O(n)$  time,  $O(1)$  extra memory.

## Solution

用quick sort来解，平均时间复杂度为 $O(n)$ 。

1. 初始为求数组中从头到尾第 $\text{nums.length} - k + 1$ 个元素（即从后往前第 $k$ 个元素，即第 $k$ 最大元素，）
2. 然后用quick sort并返回分割点的index
3. 如果该分割点正好是第 $\text{nums.length} - k + 1$ 个元素，则找到并返回；否则若该分割点比 $k$ 小，则递归地在数组 $\text{index} + 1$ —数组尾区间寻找，若该分割点比 $k$ 大，则递归地在数组头— $\text{index} - 1$ 区间寻找

代码如下：

```
class Solution {  
    /*  
     * @param k : description of k
```

```
* @param nums : array of nums
* @return: description of return
*/
public int kthLargestElement(int k, int[] nums) {
    // write your code here
    if(nums == null || nums.length == 0 || k < 1 || k > nums
.length){
        return -1;
    }

    //第k大就是从小到大排第nums.length - k + 1大
    return kthLargestHelper(nums, 0, nums.length - 1, nums.l
ength - k + 1);
}
//quick sort模板
private int kthLargestHelper(int[] nums, int l, int r, int k
){
    if(l == r){
        return nums[l];
    }

    int position = partition(nums, l, r);
    if(position + 1 == k){
        return nums[position];
    }else if(position + 1 > k){
        return kthLargestHelper(nums, l, position - 1, k);
    }else{
        return kthLargestHelper(nums, position + 1, r, k);
    }
}

private int partition(int[] nums, int l, int r){
    int start = l;
    int end = r;
    int pivot = nums[end];

    while(start < end){
        while(start < end && nums[start] <= pivot){
            start++;
        }

```

```
        nums[end] = nums[start];
        while(start < end && nums[end] > pivot){
            end--;
        }
        nums[start] = nums[end];
    }

    nums[start] = pivot;
    return start;
}

};
```

# Wiggle Sort II 507

## Question

Given an unsorted array `nums`, reorder it such that

$\text{nums}[0] < \text{nums}[1] > \text{nums}[2] < \text{nums}[3] \dots$

Notice

You may assume all input has valid answer.

Example

Given `nums = [1, 5, 1, 1, 6, 4]`, one possible answer is `[1, 4, 1, 5, 1, 6]`.

Given `nums = [1, 3, 2, 2, 3, 1]`, one possible answer is `[2, 3, 1, 3, 1, 2]`.

Challenge

Can you do it in  $O(n)$  time and/or in-place with  $O(1)$  extra space?

## Solution

这道题其实就是要找一个中间值，比中间值小的插入到偶数位上，比中间值大的插入到奇数位上。

1. 用quick select寻找中间值（即能平分`nums`中所有元素的元素的值）
2. 将`ans`的值全部设为中间值，遍历数组`nums`，将遇到的数和中间值比较，间隔地插入`ans`中
3. 若原数组长度为偶数，则为小->大->小。。。->大的形式，小的数从后往前 $(len - 2)$ 寻找位置插入，大的数从前往后 $(1)$ 寻找位置插入

若原数组长度为奇数，则为大->小->大。。。->小的形式，小的数从前往后 $(0)$ 寻找位置插入，大的数从后往前 $(len - 2)$ 寻找位置插入

代码如下：

```
public class Solution {
    /**
     * @param nums a list of integer
     * @return void
     */
    public void wiggleSort(int[] nums) {
        // Write your code here
        if(nums == null || nums.length <= 1){
            return;
        }

        int[] temp = Arrays.copyOfRange(nums, 0, nums.length);
        int avg = partition(temp, 0, nums.length - 1, nums.length / 2);

        int[] ans = new int[nums.length];
        for(int i = 0; i < nums.length; i++){
            ans[i] = avg;
        }

        int l, r;
        if(nums.length % 2 == 0){
            l = nums.length - 2;
            r = 1;
            for(int i = 0; i < nums.length; i++){
                if(nums[i] < avg){
                    ans[l] = nums[i];
                    l -= 2;
                }else if(nums[i] > avg){
                    ans[r] = nums[i];
                    r += 2;
                }
            }
        }else{
            l = 0;
            r = nums.length - 2;
            for(int i = 0; i < nums.length; i++){
                if(nums[i] < avg){
                    ans[l] = nums[i];
                    l += 2;
                }
            }
        }
    }
}
```

```

        }else if(nums[i] > avg){
            ans[r] = nums[i];
            r -= 2;
        }
    }

    for(int i = 0; i < nums.length; i++){
        nums[i] = ans[i];
    }
}

private int partition(int[] temp, int left, int right, int rank)
{
    int l = left;
    int r = right;
    int now = temp[left];
    while(l < r){
        while(l < r && temp[r] >= now){
            r--;
        }
        temp[l] = temp[r];
        while(l < r && temp[l] <= now){
            l++;
        }
        temp[r] = temp[l];
    }
    temp[l] = now;

    if(l - left < rank){
        return partition(temp, l + 1, right, rank - (l - left) - 1);
    }else if(l - left > rank){
        return partition(temp, left, r - 1, rank);
    }
    return now;
}
}

```



# Largest Number 184

## Question

Given a list of non negative integers, arrange them such that they form the largest number.

Notice

The result may be very large, so you need to return a string instead of an integer.

Example

Given [1, 20, 23, 4, 8], the largest formed number is 8423201.

Challenge

Do it in  $O(n \log n)$  time complexity.

## Solution

这道题很简单，只要利用排序把高位大的数排到前面即可。把int数转换成string，然后利用string的compareTo方法，比较s1+s2和s2+s1即可。最后只要注意把前置的0去掉。

代码如下：



```
public class Solution {
    /**
     * @param num: A list of non negative integers
     * @return: A string
     */
    class myComparator implements Comparator<Integer>{
        public int compare(Integer a, Integer b){
            String s1 = a.toString();
            String s2 = b.toString();
            return (s2 + s1).compareTo(s1 + s2);
        }
    }

    public String largestNumber(int[] num) {
        // write your code here
        String res = "";
        if(num == null || num.length == 0){
            return res;
        }

        PriorityQueue<Integer> queue = new PriorityQueue<Integer>
            >(1, new myComparator());

        for(int i = 0; i < num.length; i++){
            queue.add(num[i]);
        }

        while(!queue.isEmpty()){
            int curt = queue.poll();
            if(res.length() == 0 && curt == 0 && !queue.isEmpty(
        )){
                continue;
            }
            res = res + curt;
        }
        return res;
    }
}
```



# Sort Integers II 464

## Question

Given an integer array, sort it in ascending order. Use quick sort, merge sort, heap sort or any  $O(n\log n)$  algorithm.

Example

Given [3, 2, 1, 4, 5], return [1, 2, 3, 4, 5].

## Solution

quicksort, mergesort, heapsort。heapsort自己建heap的方法之后补上。

代码如下：

```
public class Solution {  
    /**  
     * @param A an integer array  
     * @return void  
     */  
    public void sortIntegers2(int[] A) {  
        // Write your code here  
        if(A == null || A.length <= 1){  
            return;  
        }  
        int n = A.length;  
  
        //quick sort  
        // quickSort(A, 0, n - 1);  
  
        //merge sort  
        // int[] temp = new int[A.length];  
        // mergeSort(A, 0, n - 1, temp);  
    }  
}
```

```
        //heap sort
        PriorityQueue<Integer> heap = new PriorityQueue<Integer>
();
        for(int i = 0; i < n; i++){
            heap.add(A[i]);
        }
        for(int i = 0; i < n; i++){
            A[i] = heap.poll();
        }
    }

    private void quickSort(int[] A, int start, int end){
        if(start >= end){
            return;
        }

        int pos = partition(A, start, end);
        quickSort(A, start, pos - 1);
        quickSort(A, pos + 1, end);
    }

    private int partition(int[] A, int start, int end){
        // int k = start + Math.random() * (end - start + 1);
        int pivot = A[end];
        while(start < end){
            while(start < end && A[start] <= pivot){
                start++;
            }
            A[end] = A[start];

            while(start < end && A[end] >= pivot){
                end--;
            }
            A[start] = A[end];
        }
        A[start] = pivot;
        return start;
    }
}
```

```
//merge sort
private void mergeSort(int[] A, int start, int end, int[] temp){
    if(start >= end){
        return;
    }

    int mid = (start + end) / 2;
    mergeSort(A, start, mid, temp);
    mergeSort(A, mid + 1, end, temp);
    merge(A, start, mid, end, temp);
}

private void merge(int[] A, int start, int mid, int end, int
[] temp){
    int left = start;
    int right = mid + 1;
    int index = start;

    while(left <= mid && right <= end){
        if(A[left] < A[right]){
            temp[index++] = A[left++];
        }else{
            temp[index++] = A[right++];
        }
    }

    while(left <= mid){
        temp[index++] = A[left++];
    }

    while(right <= end){
        temp[index++] = A[right++];
    }

    for(int i = start; i <= end; i++){
        A[i] = temp[i];
    }
}
}
```



# Bit Manipulation

位操作的题，一共6题。

# A + B Problem 1

## Question

Write a function that add two numbers A and B. You should not use + or any arithmetic operators.

Clarification

Are a and b both 32-bit integers?

Yes.

Can I use bit operation?

Sure you can.

Example

Given a=1 and b=2 return 3

Challenge

Of course you can just return a + b to get accepted. But Can you challenge not do it like that?

## Solution

利用异或操作 $\wedge$ 来代替+的操作。 $\wedge$ 的规则为0,0->0, 1,1->0, 0,1->1, 1,0->1，看起来和+操作非常类似，只是1+1时没有进位，因此 $\wedge$ 又被称为不进位加法。同时只要知道哪些位是需要进位的（即a和b中都为1的位置）即可。

1. 对a和b取 $\wedge$ 操作，得到不进位相加的结果
2. 对a和b取 $\&$ 操作，在a和b都为1的位置为1，其余位置为0，再将结果向左移动1位，即可得到进位的结果
3. 将1和2中结果相加
4. 重复1-3操作直到不需要进位为止



代码如下：

```
class Solution {
    /*
     * param a: The first integer
     * param b: The second integer
     * return: The sum of a and b
     */
    public int aplusb(int a, int b) {
        // write your code here, try to do it without arithmetic
        operators.
        while(b != 0){
            int _a = a ^ b;
            int _b = (a & b) << 1;
            a = _a;
            b = _b;
        }

        return a;
    }
};
```

# Count 1 in Binary 365

## Question

Count how many 1 in binary representation of a 32-bit integer.

Example

Given 32, return 1

Given 5, return 2

Given 1023, return 9

Challenge

If the integer is  $n$  bits with  $m$  1 bits. Can you do it in  $O(m)$  time?

## Solution

两种解法。

第一种比较straight forward，将每一位和1取&操作，若非0则记一次1。

第二种比较巧妙， $\text{num} - 1$ 可以将最低位的1变为0，然后和num取&操作将num中最低位的1删去并将剩下的1保留下来，如此重复直到num中的1全部变为0，变换的次数即为num中1的个数。

代码如下：

```
public class Solution {  
    /**  
     * @param num: an integer  
     * @return: an integer, the number of ones in num  
     */  
    public int countOnes(int num) {  
        // write your code here  
        int a;  
        int count = 0;  
        for(int i = 0; i < 32; i++){  
            a = 1 << i;  
            if((num & a) != 0){  
                count++;  
            }  
        }  
  
        return count;  
    }  
}
```

```
public class Solution {  
    /**  
     * @param num: an integer  
     * @return: an integer, the number of ones in num  
     */  
    public int countOnes(int num) {  
        // write your code here  
        int count = 0;  
        while(num != 0){  
            num = num & (num - 1);  
            count++;  
        }  
        return count;  
    }  
}
```



## O(1) Check Power of 2 142

### Question

Using  $O(1)$  time to check whether an integer  $n$  is a power of 2.

Example

For  $n=4$ , return true;

For  $n=5$ , return false;

Challenge

$O(1)$  time

### Solution

观察2的倍数，都是只有一位为1，因此只要检查 $n$ 中是否只有1个1即可。

$n-1$ 可以将最低位的1变为0，再和 $n$ 取&操作，若为0则说明只有刚才最低位的1个1。

代码如下：

```
class Solution {  
    /*  
     * @param n: An integer  
     * @return: True or false  
     */  
    public boolean checkPowerOf2(int n) {  
        // write your code here  
        if(n <= 0){  
            return false;  
        }  
  
        return (n & (n - 1)) == 0;  
    }  
};
```

# Flip Bits 181

## Question

Determine the number of bits required to flip if you want to convert integer  $n$  to integer  $m$ .

Notice

Both  $n$  and  $m$  are 32-bit integers.

Example

Given  $n = 31$  (11111),  $m = 14$  (01110), return 2.

## Solution

对 $a$ 和 $b$ 取 $\wedge$ 操作，如果 $a$ 和 $b$ 的某一位上不同则结果的那一位上为1，相同为0。然后只要依次看结果有几位1（不同）即可。

代码如下：

```
class Solution {
    /**
     * @param a, b: Two integer
     * @return: An integer
     */
    public static int bitSwapRequired(int a, int b) {
        // write your code here
        int count = 0;
        for(int c = a ^ b; c != 0; c = c >>> 1){
            count += c & 1;
        }

        return count;
    }
};
```





# Update Bits

## Question

Given two 32-bit numbers, N and M, and two bit positions, i and j. Write a method to set all bits between i and j in N equal to M (e.g., M becomes a substring of N located at i and starting at j)

Notice

In the function, the numbers N and M will be given in decimal, you should also return a decimal number.

Clarification You can assume that the bits j through i have enough space to fit all of M. That is, if M=10011, you can assume that there are at least 5 bits between j and i. You would not, for example, have j=3 and i=2, because M could not fully fit between bit 3 and bit 2.

Example

Given N=(10000000000)<sub>2</sub>, M=(10101)<sub>2</sub>, i=2, j=6

return N=(10001010100)<sub>2</sub>

Challenge

Minimum number of operations?

## Solution

思想很简单，就是将N的i—j的位置上的元素都变成0其他位不变，然后将M移动i位后和N相加，这样M的所有元素就被放到N的i—j位上了。

任何位**&1**之后不变，**&0**之后变为**0**，因此需要用一个mask其i—j位为0，其他位为1，然后再将mask & N即可使得N的i-j位为0，其他位保持不变。在构建mask时，当j<31时，将1移动j+1位后减去将1移动i位后的数（位减法和普通减法类似，0-1时

要向前一位借数)，这样可以得到 $i \sim j$ 全为1其他位为0，然后取反即可得到 $i \sim j$ 为0其他位为1.当 $j=31$ 时，则将1移动 $i$ 位后减去1移动0位后的数，这样可以得到 $i \sim j$ 为0， $i$ 之后其他位为1。

代码如下：

```
class Solution {
    /**
     * @param n, m: Two integer
     * @param i, j: Two bit positions
     * return: An integer
     */
    public int updateBits(int n, int m, int i, int j) {
        // write your code here
        int mask = (j < 31 ? (~(1 << (j + 1)) - (1 << i))) : ((
1 << i) - 1));
        return (m << i) + (mask & n);
    }
}
```

# Binary Representation 180

## Question

Given a (decimal - e.g. 3.72) number that is passed in as a string, return the binary representation that is passed in as a string. If the fractional part of the number can not be represented accurately in binary with at most 32 characters, return ERROR.

Example

For n = "3.72", return "ERROR".

For n = "3.5", return "11.1".

## Solution

将数字分为整数和小数两部分解决。根据“.”将n拆成两部分，注意“.”是转义字符，需要表示成“\\.”，而不能直接用“.”。corner case: n中没有“.”。

整数部分：除以2取余数直到除数为0为止，转换成二进制数。corner case: "0"或者""的情况（之前n可能是"0.x"或者".x"），直接返回"0"。

小数部分：乘以2取整数部分直到小数部分为0或者出现循环为止，转换成二进制数。corner case: "0"或者""的情况（之前n可能是"x.0"或者"x."），直接返回"0"。用一个set记录数否出现重复的数，当出现重复的数或者当超过一定位数（这里取32）还不能完全表示小数部分时，返回"ERROR"。

代码如下：

```
public class Solution {  
    /**  
     * @param n: Given a decimal number that is passed in as a string  
     * @return: A string  
     */  
    public String binaryRepresentation(String n) {
```

```
// write your code here
if(n == null || n.length() == 0){
    return "0";
}

if(n.indexOf('.') == -1){
    return parseInteger(n);
}

String[] num = n.split("\\.");
String f = parseFloat(num[1]);
if(f.equals("ERROR")){
    return "ERROR";
}

if(f.equals("") || f.equals("0")){
    return parseInteger(num[0]);
}

return parseInteger(num[0]) + "." + f;
}

private String parseInteger(String s){
    int d = Integer.parseInt(s);
    if(s.equals("") || s.equals("0")){
        return "0";
    }
    String res = "";
    while(d != 0){
        res = Integer.toString(d % 2) + res;
        d /= 2;
    }
    return res;
}

private String parseFloat(String s){
    double d = Double.parseDouble("0." + s);
    if(s.equals("") || s.equals("0")){
        return "0";
    }
}
```

```
String res = "";
HashSet<Double> set = new HashSet<Double>();
while(d != 0){
    //出现循环
    if(res.length() > 32 || set.contains(d)){
        return "ERROR";
    }
    set.add(d);
    d = d * 2;
    if(d >= 1){
        res += "1";
        d = d - 1;
    }else{
        res += "0";
    }
}
return res;
}
```

# Divide Two Integers 414

## Question

Divide two integers without using multiplication, division and mod operator.

If it is overflow, return 2147483647

Example

Given dividend = 100 and divisor = 9, return 11.

## Solution

这道题要求不用乘，除和模操作完成除法操作。

1. 基本思想是不断地减掉除数，直到为0为止。但是这样会太慢。
2. 我们可以使用2分法来加速这个过程。不断对除数 \* 2，直到它恰好比被除数小（即再乘2就会比被除数大）为止。加倍的同时，也记录下cnt，将被除数减掉加倍后的值，并且结果+cnt。因为是2倍地加大，所以速度会很快，指数级的速度。

但是因为不能用乘除操作，因此只能用位运算的方法，即左移1位 = \* 2。

首先讨论一些边界情况：

1. 如果除数为0，则被除数为正时，结果为正无穷，否则为负无穷
2. 如果被除数为0，则返回0
3. 如果被除数为正无穷除数为1或者被除数为负无穷除数为-1，则返回正无穷
4. 如果被除数为正无穷除数为-1或者被除数为负无穷除数为1，则返回负无穷

记录一下结果的符号（即除数和被除数同号为+，异号为-），然后对除数和被除数取绝对值。注意：因为int最大2147483647,最小-2147483648，最小值直接abs后会溢出，所以要先讲被除数与除数转换为long之后再取绝对值，以防出现溢出的情况。

每次将**b**左移一定的位数，比如*i*位，使得**b**恰好小于**a**（即**b**<**a**），此时相当于**b**  $2^i$ ，然后再对剩下的数（即**a**-**b**  $2^i$ ）做刚才的操作，直到剩下的数小于**b**为止。看**b**总共乘以了多少*power of 2*，将其加和即可。

代码如下：

```
public class Solution {
    /**
     * @param dividend the dividend
     * @param divisor the divisor
     * @return the result
     */
    public int divide(int dividend, int divisor) {
        // Write your code here
        if(divisor == 0){
            return dividend >= 0 ? Integer.MAX_VALUE : Integer.M
IN_VALUE;
        }

        if(dividend == 0){
            return 0;
        }

        if((dividend == Integer.MAX_VALUE && divisor == 1) || (d
ividend == Integer.MIN_VALUE && divisor == -1)){
            return Integer.MAX_VALUE;
        }

        if((dividend == Integer.MIN_VALUE && divisor == 1) || (d
ividend == Integer.MAX_VALUE && divisor == -1)){
            return Integer.MIN_VALUE;
        }

        boolean signal = ((dividend > 0 && divisor > 0) || (divi
dend < 0 && divisor < 0))? true : false;

        //在这里必须先取long再abs，否则int的最小值abs后也是原值(因为int
        最大2147483647, 最小-2147483648, 最小值abs后会溢出)
        long a = Math.abs((long)dividend);
        long b = Math.abs((long)divisor);
```

```
    int result = 0;
    while(a >= b){
        int shift = 0;
        while(a >= (b << shift)){
            shift++;
        }
        shift--;
        a -= b << shift;
        result += 1 << shift;
    }

    return signal? result : -result;
}
```



# Gray Code 411

## Question

The gray code is a binary numeral system where two successive values differ in only one bit.

Given a non-negative integer  $n$  representing the total number of bits in the code, find the sequence of gray code. A gray code sequence must begin with 0 and with cover all  $2^n$  integers.

Notice

For a given  $n$ , a gray code sequence is not uniquely defined.

[0,2,3,1] is also a valid gray code sequence according to the above definition.

Example

Given  $n = 2$ , return [0,1,3,2]. Its gray code sequence is:

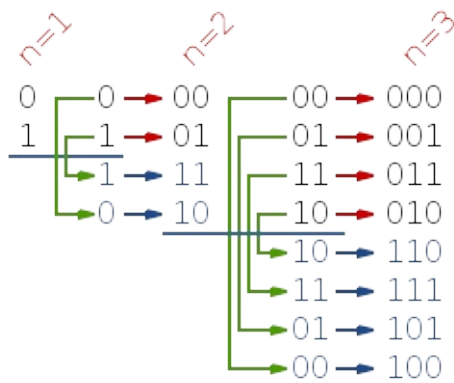
00 - 0 01 - 1 11 - 3 10 - 2

Challenge

$O(2^n)$  time.

## Solution

第一种方法：参考维基百科上关于格雷码的性质，有一条是说镜面排列的， $n$ 位元的格雷码可以从 $n-1$ 位元的格雷码以上下镜射后加上新位元的方式快速的得到，如下图所示一般。



根据这条性质，求 $n$ 位的gray code，可以根据 $n-1$ 位的gray code来求。只要将 $n-1$ 位的gray code的答案在前面加一位0以及 $n-1$ 位的gray code的答案reverse之后全部在前面加一位1即可。

第二种方法：用set来保存已经产生的结果，我们从0开始，遍历其二进制每一位，对其取反，然后看其是否在set中出现过，如果没有，我们将其加入set和结果res中，然后再对这个数的每一位进行遍历，以此类推就可以找出所有的格雷码了。

其他更多方法（比如非递归）见：[gray code](#)

代码如下：

mirror version

```

public class Solution {
    /**
     * @param n a number
     * @return Gray code
     */
    public ArrayList<Integer> grayCode(int n) {
        // Write your code here
        ArrayList<Integer> res = new ArrayList<Integer>();
        if(n <= 1){
            for(int i = 0; i <= n; i++){
                res.add(i);
            }
            return res;
        }
        // res = grayCode(n - 1);
        // ArrayList<Integer> reversedRes = reverse(res);
        // //只考虑增加新的一位为1时的情况，因为新的一位为0时不改变值所以
        //不用加
        // int x = 1 << (n - 1);
        // for(int i = 0; i < reversedRes.size(); i++){
        //     res.add(reversedRes.get(i) + x);
        // }
        // return res;

        //set version
    }

    private ArrayList<Integer> reverse(ArrayList<Integer> res){
        ArrayList<Integer> reversed = new ArrayList<Integer>();
        for(int i = res.size() - 1; i >= 0; i--){
            reversed.add(res.get(i));
        }
        return reversed;
    }
}

```

Hashset version

```

public class Solution {

```

```
/**
 * @param n a number
 * @return Gray code
 */
public ArrayList<Integer> grayCode(int n) {
    // Write your code here
    ArrayList<Integer> res = new ArrayList<Integer>();
    if(n <= 1){
        for(int i = 0; i <= n; i++){
            res.add(i);
        }
        return res;
    }

    HashSet<Integer> set = new HashSet<Integer>();
    helper(n, set, 0, res);
    return res;
}

private void helper(int n, HashSet<Integer> set, int curt, ArrayList<Integer> res){
    if(!set.contains(curt)){
        set.add(curt);
        res.add(curt);
    }

    for(int i = 0; i < n; i++){
        int temp = curt;
        //如果当前位为0，则变为1
        if((temp & (1 << i)) == 0){
            temp |= 1 << i;
        }else{
            //如果当前位为1，则变为0，因为是&操作，为了保证其他位不变，必须使其他位为1
            temp &= ~(1 << i);
        }
        if(set.contains(temp)){
            continue;
        }
        helper(n, set, temp, res);
    }
}
```

```
    }  
  }  
}
```

# Bitwise AND of Numbers Range (LeetCode) 201

## Question

Given a range  $[m, n]$  where  $0 \leq m \leq n \leq 2147483647$ , return the bitwise AND of all numbers in this range, inclusive.

For example, given the range  $[5, 7]$ , you should return 4.

## Solution

这道题思路如下

1. last bit of (odd number & even number) is 0.
2. when  $m \neq n$ , There is at least an odd number and an even number, so the last bit position result is 0.
3. Move  $m$  and  $n$  right a position.

Keep doing step 1,2,3 until  $m$  equal to  $n$ , use a factor to record the iteration time.

简单来说就是 $m$ 和 $n$ 的相同部分&得到的都是1，不同部分&都是0。当 $m$ 和 $n$ 不同时，相同部分在高位，因此每次同时向右移动1为去掉最低位的不同部分，直到剩下相同部分为止。用一个值记录移动了多少次，再将相同部分向左移动相同次数即可。

代码如下：

```
public class Solution {  
  
    public int rangeBitwiseAnd(int m, int n) {  
  
        int step = 0;  
  
        while (m != n) {  
  
            m >>= 1;  
  
            n >>= 1;  
  
            step++;  
  
        }  
  
        return m << step;  
    }  
}
```

```
}
```

```
return m << step;
```

```
}
```

```
}
```

# Permutation



# Permutation Index 197

## Question

Given a permutation which contains no repeated number, find its index in all the permutations of these numbers, which are ordered in lexicographical order. The index begins at 1.

Example

Given [1,2,4], return 1.

## Solution

这道题因为没有重复元素，所以比较简单。主要思想为找出当前位元素的index，然后将各位加起来即可。

然后顺序遍历A中的元素，求当前元素的index：小于当前元素的可用元素的数量 \* 之后元素数量的阶乘。然后要更新map，因为当前元素在这次已经使用，所以在map里的比当前元素大的元素的value要-1。

代码如下：

```
public class Solution {  
    /**  
     * @param A an integer array  
     * @return a long integer  
     */  
    public long permutationIndex(int[] A) {  
        // Write your code here  
        if(A == null || A.length == 0){  
            return 0;  
        }  
  
        int n = A.length;  
        //为了防止操作改变A的顺序，要复制一份来排序  
        int[] array = Arrays.copyOf(A, n);
```

```
Arrays.sort(array);
//将有多少个元素小于当前元素的信息存在map中
HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
for(int i = 0; i < n; i++){
    map.put(array[i], i);
}

//index start from 1
long sum = 1;
for(int i = 0; i < n; i++){
    sum += map.get(A[i]) * factor(n - 1 - i);
    updateMap(map, A[i]);
}

return sum;
}

private long factor(int n){
    long now = 1;
    for(int i = 1; i <= n; i++){
        now *= (long) i;
    }
    return now;
}

private void updateMap(HashMap<Integer, Integer> map, int a)
{
    for(int key : map.keySet()){
        if(a < key){
            map.put(key, map.get(key) - 1);
        }
    }
}
}
```

# Permutation Index II 198

## Question

Given a permutation which may contain repeated numbers, find its index in all the permutations of these numbers, which are ordered in lexicographical order. The index begins at 1.

Example

Given the permutation [1, 4, 2, 2], return 3.

## Solution

这道题和Permutation Index I思想一样，计算每一位上数字是该位上第几个排列，再将每一位结果加和即可。只是这道题有重复元素，有无重复元素最大的区别在于原来的1!, 2!, 3!...等需要除以重复元素个数的阶乘。按照数字从低位到高位进行计算。每遇到一个重复的数字就更新重复元素个数的阶乘的值。

1. 从后往前遍历数组，用一个hashmap来记录重复元素个数。若新来的数不是重复元素，则加入hashmap，否则将重复元素个数+1，同时更新重复元素个数的阶乘。
2. 比较当前位和其后面位的数，计算当前位是第几大的数count
3. 当前位的index为：2的结果count \* 其后面位数的阶乘/重复数个数的阶乘。将当前位计入阶乘，重复1-3计算前一位。

注意：1.题目说index从1开始算。2.要用long来保存index，fact和multFact，用int有可能超过范围

代码如下：

```
public class Solution {  
    /**  
     * @param A an integer array  
     * @return a long integer
```

```
    */
    public long permutationIndexII(int[] A) {
        // Write your code here
        if(A == null || A.length == 0){
            return 0;
        }

        //用int可能越界
        long index = 1; //index start from 1
        long fact = 1;
        long multiFact = 1;
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();

        for(int i = A.length - 1; i >= 0; i--){
            if(!map.containsKey(A[i])){
                map.put(A[i], 1);
            }else{
                map.put(A[i], map.get(A[i]) + 1);
                multiFact *= map.get(A[i]);
            }

            int count = 0;
            for(int j = i + 1; j < A.length; j++){
                if(A[i] > A[j]){
                    count++;
                }
            }

            index += count * fact / multiFact;
            fact *= A.length - i;
        }

        return index;
    }
}
```

# Next Permutation 52

## Question

Given a list of integers, which denote a permutation.

Find the next permutation in ascending order.

Notice

The list may contains duplicate integers.

Example

For [1,3,2,3], the next permutation is [1,3,3,2]

For [4,3,2,1], the next permutation is [1,2,3,4]

## Solution

下一个permutation就是使得整个数列增加的值最小的变化（除了当前数列为递减数列，则下一个permutation为递增数列）。为了使得下一个数列与当前数列相比增加值最小，则要使增加的位置越低位（越靠右）越好，同时增加增加的值越小越好，因此从后往前遍历每一位，寻找第一个比当前位小的数，找到一个最靠右的第一个小的数，如果有两个数的第一个小的数的位置相同，则选择较小的那个数（增加值小）。遍历完成后，将那个数和第一个小的数交换位置，同时对第一个原来第一个小的数的位置之后的所有数递增排序即可。如果遍历完成后没有找到第一个小的数，则说明当前数列递减，则对整个数列递增排序即可。

代码如下：

```
public class Solution {  
    /**  
     * @param nums: an array of integers  
     * @return: return nothing (void), do not return anything, modify nums in-place instead  
     */  
}
```

```

public int[] nextPermutation(int[] nums) {
    // write your code here
    if(nums == null || nums.length == 0){
        return nums;
    }

    int pos = -1;
    int index = -1;
    //从后往前寻找每位上的数遇到的第一个比自己小的数，记录下第一个小的
    数的位置以及当前数的值，选择位置最大（最靠右）的第一个大的数和最小的当前数（
    即增加值最小）
    for(int i = nums.length - 1; i > 0; i--){
        if(i <= pos){
            break;
        }

        for(int j = i - 1; j >= 0; j--){
            if(nums[i] > nums[j]){
                if(pos > j){
                    break;
                }else if(pos == j && nums[i] >= nums[index])
            {
                break;
            }else{
                pos = j;
                index = i;
            }
        }
    }

    //说明当前数组减，则next permutation为递增数组，对整个数组排序即可
    if(pos == -1){
        Arrays.sort(nums);
        int i = 0;
        //防止出现0开头的情况
        while(i < nums.length && nums[i] == 0){
            i++;
        }
        int temp = nums[0];

```

```
        nums[0] = nums[i];
        nums[i] = temp;
    }else{
        //交换两个数，同时对改变值的位置之后的所有数排序
        int temp = nums[pos];
        nums[pos] = nums[index];
        nums[index] = temp;
        Arrays.sort(nums, pos + 1, nums.length);
    }
    return nums;
}
}
```

# Next Permutation II 190

## Question

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

### Example

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

1,2,3 → 1,3,2

3,2,1 → 1,2,3

1,1,5 → 1,5,1

### Challenge

The replacement must be in-place, do not allocate extra memory.

## Solution

思想基本和I一样。

首先找到需要变大的那一位。因为求的是next permutation，所以增大位数要尽量靠右边同时增大的数值要尽量小。具体方法如下：

1) 从数组尾部开始，依次找每一位数离它最近的比它小的数的位置（即要变大的位置），记录下这个数和离它最近的比它小的数的index。每一位数得到的结果都和之前保存的结果做比较，保留index最大的要变大的位置和找到这个位置的数的位置，当要变大的位置相同时，保留数值较小的找到这个位置的数的位置。

2) 遍历完整个数组后，根据得到的结果，将要变大位置的数和找到这个位置的数交换位置，并返回变大的位置



3) 将数组从变大的位置的后一位开始从小到大排序即可。这里因为要求inplace所以用了bubble sort，若可以用额外空间还能用更快的排序方法。

代码如下：

```
public class Solution {
    /**
     * @param nums: an array of integers
     * @return: return nothing (void), do not return anything, modify nums in-place instead
     */
    public void nextPermutation(int[] nums) {
        // write your code here
        if(nums == null || nums.length == 0){
            return;
        }

        int index = findMax(nums);

        bubbleSort(nums, index + 1, nums.length - 1);
    }

    private int findMax(int[] nums){
        int[] temp = new int[]{-1, -1};
        for(int i = nums.length - 1; i >= 0; i--){
            for(int j = i - 1; j >= 0; j--){
                if(nums[i] > nums[j]){
                    if(j > temp[0]){
                        temp[0] = j;
                        temp[1] = i;
                    }else if(j == temp[0]){
                        if(nums[i] < nums[temp[1]]){
                            temp[1] = i;
                        }
                    }
                }
            }
            break;
        }
    }

    if(temp[0] == -1){
```

```
        return -1;
    }
    int a = nums[temp[0]];
    nums[temp[0]] = nums[temp[1]];
    nums[temp[1]] = a;
    return temp[0];
}

private void bubbleSort(int[] nums, int start, int end){
    for(int i = start; i < end; i++){
        for(int j = start; j < end; j++){
            if(nums[j] > nums[j + 1]){
                int temp = nums[j];
                nums[j] = nums[j + 1];
                nums[j + 1] = temp;
            }
        }
    }
}
```

# Previous Permutation 51

## Question

Given a list of integers, which denote a permutation.

Find the previous permutation in ascending order.

Notice

The list may contains duplicate integers.

Example

For [1,3,2,3], the previous permutation is [1,2,3,3]

For [1,2,3,4], the previous permutation is [4,3,2,1]

## Solution

这题就是next permutation的逆过程。首先找到从后往前遍历，直到找到一个数（假设位置为*i*）比之它前一位数小（即该位到数组的最后是一个递增序列），然后将*i*到数组最后的所有数按递减排列。然后在这个递增数列里面找到第一个比*i-1*位置上数小的数，交换两个数的位置即可。如果*i*=1则说明之前的整个数组都是递增序列，现在整个数组变为递减数列，正好是previous permutation，因此不用再做任何其他操作。

代码如下：

```
public class Solution {  
    /**  
     * @param nums: A list of integers  
     * @return: A list of integers that's previous permuation  
     */  
    public ArrayList<Integer> previousPermutation(ArrayList<Integer> nums) {  
        // write your code  
        if(nums == null || nums.size() == 0){
```

```
        return nums;
    }

    int length = nums.size();
    int i = length - 1;
    while(i > 0 && nums.get(i) >= nums.get(i - 1)){
        i--;
    }
    swapList(nums, i, length - 1);

    if(i != 0){
        int j = i;
        while(nums.get(i - 1) <= nums.get(j)){
            j++;
        }
        swapItem(nums, i - 1, j);
    }

    return nums;
}

private void swapList(ArrayList<Integer> nums, int start, int end){
    while(start < end){
        swapItem(nums, start, end);
        start++;
        end--;
    }
}

private void swapItem(ArrayList<Integer> nums, int i, int j)
{
    int temp = nums.get(i);
    nums.set(i, nums.get(j));
    nums.set(j, temp);
}
}
```



# Greedy

# Delete Digits 182

## Question

Given string A representative a positive integer which has N digits, remove any k digits of the number, the remaining digits are arranged according to the original order to become a new positive integer.

Find the smallest integer after remove k digits.

$N \leq 240$  and  $k \leq N$ ,

Example

Given an integer A = "178542", k = 4

return a string "12"

## Solution

要让一个数尽量小，那么就要把小的数字尽量放到前面，如果前面有比它大的数字，那么就到把在它前面且比它大的数字都要删除掉，直到已经删掉k个数字，所以最后留下的是一个递增数列。同时要考虑一些特殊情况，比如前置0要去掉，以及如果遍历一遍之后发现删去的数不足k个，则删去最后的k-count个数。

代码如下：

```
public class Solution {  
    /**  
     * @param A: A positive integer which has N digits, A is a string.  
     * @param k: Remove k digits.  
     * @return: A string  
     */  
    public String DeleteDigits(String A, int k) {  
        // write your code here  
        String res = "";
```

```
    if(A == null || A.length() == 0){
        return res;
    }
    //为了使得剩下的数最小，要尽可能地把高位的大数删掉
    Stack<Character> stack = new Stack<Character>();
    int count = 0;
    for(int i = 0; i < A.length(); i++){
        char curt = A.charAt(i);
        while(!stack.isEmpty() && stack.peek() > curt && count < k){
            stack.pop();
            count++;
        }
        stack.push(curt);
    }

    Stack<Character> temp = new Stack<Character>();
    while(!stack.isEmpty()){
        temp.push(stack.pop());
    }
    while(!temp.isEmpty()){
        res = res + temp.pop();
    }

    //count < k case
    if(count < k){
        int num = k - count;
        res = res.substring(0, res.length() - num);
    }

    //front 0 case
    int i = 0;
    while(i < res.length() && res.charAt(i) == '0'){
        i++;
    }
    if(i == res.length()){
        res = "0";
    }else{
        res = res.substring(i);
    }
}
```



```
        return res;
    }
}
```

优化版本：

```
public class Solution {
    /**
     * @param A: A positive integer which has N digits, A is a string.
     * @param k: Remove k digits.
     * @return: A string
     */
    public String DeleteDigits(String A, int k) {
        // write your code here
        StringBuilder sb = new StringBuilder(A);
        int i, j;
        for (i = 0; i < k; i++) {
            for (j = 0; j < sb.length() - 1
                && sb.charAt(j) <= sb.charAt(j + 1); j++) {
            }
            sb.delete(j, j + 1);
        }
        while (sb.length() > 1 && sb.charAt(0) == '0') {
            sb.delete(0, 1);
        }
        return sb.toString();
    }
}
```

# Find the Missing Number 196

## Question

Given an array contains N numbers of 0 .. N, find which number doesn't exist in the array.

Example

Given  $N = 3$  and the array  $[0, 1, 3]$ , return 2.

Challenge

Do it in-place with  $O(1)$  extra memory and  $O(n)$  time.

## Solution

这道题和first missing positive number思想一样。

遍历数组，看当前位置上的数和其下标是否一致，若不一致，则将该数和正确下标位置上的数交换，然后继续检查新来的数，直到满足条件为止。

最后重新遍历数组，找到第一个下标和数不一致的下标返回即可。若全部满足，则返回下标最大值+1，即数组长度。

代码如下：

```
public class Solution {  
    /**  
     * @param nums: an array of integers  
     * @return: an integer  
     */  
    public int findMissing(int[] nums) {  
        // write your code here  
        //和first missing positive number思想一样。遍历数组，看当前位置上的数和其下标是否一致，若不一致，则将该数和正确下标位置上的数交换，然后继续检查新来的数，直到满足条件为止。  
        if(nums == null || nums.length == 0){  
            return 0;  
        }  
  
        for(int i = 0; i < nums.length; i++){  
            while(nums[i] != i && nums[i] < nums.length){  
                //index为num[i]因该被放到的位置  
                int index = nums[i];  
                //若要放到的位置上已经有正确的数则不交换  
                if(nums[index] == nums[i]){  
                    break;  
                }  
                int temp = nums[i];  
                nums[i] = nums[index];  
                nums[index] = temp;  
            }  
        }  
  
        for(int i = 0; i < nums.length; i++){  
            if(nums[i] != i){  
                return i;  
            }  
        }  
  
        return nums.length;  
    }  
}
```



# Gas Station 187

## Question

There are  $N$  gas stations along a circular route, where the amount of gas at station  $i$  is  $gas[i]$ .

You have a car with an unlimited gas tank and it costs  $cost[i]$  of gas to travel from station  $i$  to its next station  $(i+1)$ . You begin the journey with an empty tank at one of the gas stations.

Return the starting gas station's index if you can travel around the circuit once, otherwise return  $-1$ .

Notice

The solution is guaranteed to be unique.

Example

Given 4 gas stations with  $gas[i]=[1,1,3,1]$ , and the  $cost[i]=[2,2,1,1]$ . The starting gas station's index is 2.

Challenge

$O(n)$  time and  $O(1)$  extra space

## Solution

这道题只要看 $gas$ 和是否小于 $cost$ 和，若小于则一定没有解，反之则一定有解，此时只要找到开头（即当前 $gas$ 的值加上之前剩下的 $gas$ 的值大于等于当前 $cost$ 的值，即能够到达下一个 $station$ ），看其到数组最后一个 $station$ 中间的每个点是否都满足条件（当前 $gas$ 的值加上之前剩下的 $gas$ 的值大于等于当前 $cost$ 的值），若满足则返回此开头，否则从不满足的点的下一个点开始重新寻找开头。该问题属于贪心算法。

代码如下：

```
public class Solution {
    /**
     * @param gas: an array of integers
     * @param cost: an array of integers
     * @return: an integer
     */
    //若gas和小于cost和则一定没有解，反之则一定有解
    public int canCompleteCircuit(int[] gas, int[] cost) {
        // write your code here
        if(gas == null || gas.length == 0 || cost == null || cost.length == 0 || gas.length != cost.length){
            return -1;
        }

        int n = gas.length;
        int sumGas = 0;
        int sumCost = 0;
        for(int i = 0; i < n; i++){
            sumGas += gas[i];
            sumCost += cost[i];
        }
        if(sumGas < sumCost){
            return -1;
        }

        int start = 0;
        int diff = 0;
        for(int i = 0; i < n; i++){
            if(diff + gas[i] < cost[i]){
                start = i + 1;
                diff = 0;
            }else{
                diff += gas[i] - cost[i];
            }
        }

        return start;
    }
}
```



# String



# Roman to Integer 419

## Question

Given a roman numeral, convert it to an integer.

The answer is guaranteed to be within the range from 1 to 3999.

Symbol I V X L C D M

Value 1 5 10 50 100 500 1,000

Example

IV -> 4

XII -> 12

XXI -> 21

XCIX -> 99

## Solution

罗马数字和阿拉伯数字转化，用一个map来保存之前的对应关系。同时要考虑几种特殊情况比如IV,IX, XL, XC, CD, CM。

方法一：从后往前把罗马数字表示的数加起来，只要考虑两种情况：

1. 如果当前位比后面一位小，则表示出现上面说的那几个特殊情况，因此要将当前位表示的数减去
2. 如果当前位比后面一位大或者相等，则直接加上当前位表示的数即可

方法二：我们也可以每次跟前面的数字比较，如果小于等于前面的数字，我们先加上当前的数字，如果大于前面的数字，我们加上当前的数字减去二倍前面的数字，这样可以把在上一个循环多加数减掉。代码类似与第一种方法。

代码如下：

```
public class Solution {
    /**
     * @param s Roman representation
     * @return an integer
     */
    public int romanToInt(String s) {
        // Write your code here
        if(s == null || s.length() == 0){
            return 0;
        }

        int n = s.length();
        HashMap<Character, Integer> map = new HashMap<Character,
Integer>();
        map.put('I', 1);
        map.put('V', 5);
        map.put('X', 10);
        map.put('L', 50);
        map.put('C', 100);
        map.put('D', 500);
        map.put('M', 1000);

        int res = map.get(s.charAt(n - 1));
        for(int i = n - 2; i >= 0; i--){
            if(map.get(s.charAt(i + 1)) <= map.get(s.charAt(i)))
{
                res += map.get(s.charAt(i));
            }else{
                res -= map.get(s.charAt(i));
            }
        }

        return res;
    }
}
```

# Integer to Roman 418

## Question

Given an integer, convert it to a roman numeral.

The number is guaranteed to be within the range from 1 to 3999.

Example

4 -> IV

12 -> XII

21 -> XXI

99 -> XCIX

## Solution

与Roman to Integer类似，这题是阿拉伯数字与罗马数字转换，用两个一一对应的array来分别存储罗马数字和阿拉伯数字。

根据转化的规则，从前往后遍历数字的每一位，讨论4种情况即可：

M位千位对应的罗马数字，C为百位对应的罗马数字，X为十位对应的罗马数字，I位个位对应的罗马数字

1. 如果当前位数字 $v < 4$ ，则说明该位对应的罗马数字重复出现 $v$ 次
2. 如果当前位数字 $v = 4$ ，则说明对应的罗马数字由当前位对应的罗马数字和其前一位罗马数字组成（如IV）
3. 如果当前位数字 $v > 4$  &&  $v < 9$ ，则说明对应的罗马数字由其前一位罗马数字和 $(v-5)$ 个当前位对应的罗马数字组成(如VI)
4. 如果当前位数字 $v = 9$ ，则说明该位对应的罗马数字由当前位和其前二位罗马数字组成（如IX）

代码如下：

```
public class Solution {
    /**
     * @param n The integer
     * @return Roman representation
     */
    public String intToRoman(int n) {
        // Write your code here
        String res = "";
        if(n < 1){
            return res;
        }
        //M位千位对应的罗马数字，C为百位对应的罗马数字，X为十位对应的罗马
        数字，I位个位对应的罗马数字
        char[] roman = {'M', 'D', 'C', 'L', 'X', 'V', 'I'};
        int[] value = {1000, 500, 100, 50, 10, 5, 1};

        for(int i = 0; i < 7; i += 2){
            int v = n / value[i];
            if(v < 4){
                for(int j = 1; j <= v; j++){
                    res += roman[i];
                }
            }else if(v == 4){
                res = res + roman[i] + roman[i - 1];
            }else if(v > 4 && v < 9){
                res += roman[i - 1];
                for(int j = 6; j <= v; j++){
                    res += roman[i];
                }
            }else if(v == 9){
                res = res + roman[i] + roman[i - 2];
            }
            n %= value[i];
        }

        return res;
    }
}
```



# Length of Last Word 422

## Question

Given a string `s` consists of upper/lower-case alphabets and empty space characters ' ', return the length of last word in the string.

If the last word does not exist, return 0.

Notice

A word is defined as a character sequence consists of non-space characters only.

Example

Given `s = "Hello World"`, return 5.

## Solution

方法一：可以直接对`string`去掉首尾的空格，然后根据空格分割字符串，取最后的元素即可。

方法二：从后往前遍历，忽略遇见第一个非空格字符之前的所有空格，在遇见第一个非空格字符之后再次遇见空格字符则`break`。

代码如下：

方法一：

```
public class Solution {  
    /**  
     * @param s A string  
     * @return the length of last word  
     */  
    public int lengthOfLastWord(String s) {  
        // Write your code here  
        if(s == null || s.length() == 0){  
            return 0;  
        }  
  
        String[] str = s.trim().split(" ");  
  
        return str[str.length - 1].length();  
    }  
}
```

方法二：

```
public class Solution {  
    /**  
     * @param s A string  
     * @return the length of last word  
     */  
    public int lengthOfLastWord(String s) {  
        // Write your code here  
        if(s == null || s.length() == 0){  
            return 0;  
        }  
  
        int n = s.length();  
        int length = 0;  
        for(int i = n - 1; i >= 0; i--){  
            if(length == 0){  
                if(s.charAt(i) == ' '){  
                    continue;  
                }else{  
                    length++;  
                }  
            }else{  
                if(s.charAt(i) == ' '){  
                    break;  
                }else{  
                    length++;  
                }  
            }  
        }  
        return length;  
    }  
}
```



# Space Replacement 212

## Question

Write a method to replace all spaces in a string with %20. The string is given in a characters array, you can assume it has enough space for replacement and you are given the true length of the string.

Your code should also return the new length of the string after replacement.

Notice

If you are using Java or Python, please use characters array instead of string.

Example

Given "Mr John Smith", length = 13.

The string after replacement should be "Mr%20John%20Smith", you need to change the string in-place and return the new length 17.

Challenge

Do it in-place.

## Solution

这道题要求将“ ”替换成"%20"。

首先找到一共有多少个" "，将“ ”替换成"%20"，则每个空格比原来多2个字符，因此新的字符串长度为原字符串长度+2 \* 空格数目。

然后从后往前遍历原数组，新数组也从后往前加入新元素。若遇到的字符是" "，则将"0","2","%"依次加入新数组（从后往前），若不是" "，则直接将遇到的字符加入新数组。在遇到最后一个" "之前一定不会出现新数组加入元素的左边界快于原数组的情况，直到最后一个" "两个的左边界才会重合。

代码如下：

```
public class Solution {
    /**
     * @param string: An array of Char
     * @param length: The true length of the string
     * @return: The true length of new string
     */
    public int replaceBlank(char[] string, int length) {
        // Write your code here
        int count = 0;
        for(int i = 0; i < length; i++){
            if(string[i] == ' '){
                count++;
            }
        }
        int newLen = length + 2 * count;

        int j = 1;
        for(int i = length - 1; i >= 0; i--){
            if(string[i] != ' '){
                string[newLen - j] = string[i];
                j++;
            }else{
                string[newLen - j] = '0';
                j++;
                string[newLen - j] = '2';
                j++;
                string[newLen - j] = '%';
                j++;
            }
        }

        return newLen;
    }
}
```

# Two Strings Are Anagrams 158

## Question

Write a method `anagram(s,t)` to decide if two strings are anagrams or not.

Clarification

What is Anagram?

- Two strings are anagram if they can be the same after change the order of characters.

Example

Given `s = "abcd"`, `t = "dcab"`, return `true`.

Given `s = "ab"`, `t = "ab"`, return `true`.

Given `s = "ab"`, `t = "ac"`, return `false`.

Challenge

$O(n)$  time,  $O(1)$  extra space

## Solution

思想很简单，用256长度的字符数组存所有字符的个数。先遍历`s`，将`s`中每个字符及其个数记录下来。然后再遍历`t`，每遇到一个字符，就去字符数组的对应位置-1，如果出现负数，则返回`false`。

要先讨论几种corner case：是否为null，长度是否相等

代码如下：

```
public class Solution {
    /**
     * @param s: The first string
     * @param b: The second string
     * @return true or false
     */
    public boolean anagram(String s, String t) {
        // write your code here
        if(s == null && t == null){
            return true;
        }

        if(s == null || t == null){
            return false;
        }

        if(s.length() != t.length()){
            return false;
        }

        //有256个character
        int[] count = new int[256];
        for (int i = 0; i < s.length(); i++) {
            count[(int) s.charAt(i)]++;
        }
        for (int i = 0; i < t.length(); i++) {
            count[(int) t.charAt(i)]--;
            if (count[(int) t.charAt(i)] < 0) {
                return false;
            }
        }
        return true;
    }
};
```

# String to Integer (Leetcode 8)

## Question

Implement atoi to convert a string to an integer.

Hint: Carefully consider all possible input cases. If you want a challenge, please do not see below and ask yourself what are the possible input cases.

Notes: It is intended for this problem to be specified vaguely (ie, no given input specs). You are responsible to gather all the input requirements up front.

Update (2015-02-10): The signature of the C++ function had been updated. If you still see your function signature accepts a `const char *` argument, please click the reload button to reset your code definition.

spoilers alert... click to show requirements for atoi.

Requirements for atoi: The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in `str` is not a valid integral number, or if no such sequence exists because either `str` is empty or it contains only whitespace characters, no conversion is performed.

If no valid conversion could be performed, a zero value is returned. If the correct value is out of the range of representable values, `INT_MAX` (2147483647) or `INT_MIN` (-2147483648) is returned.

## Solution

这道题不难，就是将string转化成整数。但是要考虑的特殊情况。

1. 首先要将开始的所有空白字符去掉
2. 记录正负号。如果第一个不是空白的字符为“-”，则记录为负，否则记录为正
3. 然后一位一位加起来即可。为了防止越界，用double来记结果
4. 考虑最大值和最小值的情况

代码如下：

```
public class Solution {
    public int myAtoi(String str) {
        if(str == null || str.length() == 0){
            return 0;
        }

        str = str.trim();

        int start = 0;
        boolean isNegative = false;
        if(str.charAt(0) == '-'){
            isNegative = true;
            start++;
        }else if(str.charAt(0) == '+'){
            start++;
        }

        double res = 0;
        while(start < str.length() && Character.isDigit(str.charAt(start))){
            int temp = str.charAt(start++) - '0';
            res = res * 10 + temp;
        }

        if(isNegative){
            res = -res;
        }

        if(isNegative){
            return (int)Math.max(Integer.MIN_VALUE, res);
        }else{
            return (int)Math.min(Integer.MAX_VALUE, res);
        }
    }
}
```

## strStr 13

### Question

For a given source string and a target string, you should output the first index(from 0) of target string in source string.

If target does not exist in source, just return -1.

Clarification

Do I need to implement KMP Algorithm in a real interview?

Not necessary. When you meet this problem in a real interview, the interviewer may just want to test your basic implementation ability. But make sure your confirm with the interviewer first.

Example

If source = "source" and target = "target", return -1.

If source = "abcdabcdefg" and target = "bcd", return 1.

Challenge

$O(n^2)$  is acceptable. Can you implement an  $O(n)$  algorithm? (hint: KMP)

### Solution

这道题很简单，可以遍历第一个string的字符，看以该字符为起点的字符串能否和第二个string匹配，如果不能，则每次向下一位继续寻找。这样的时间复杂度是  $O(mn)$ 。

而这种字符串匹配的题目经典的方法是用KMP(Knuth-Morris-Pratt)算法，这种算法的时间复杂度是  $O(m+n)$ 。具体分析可以参考【[经典算法](#)】——[KMP](#)，深入讲解[next数组的求解](#)。



这里再额外解释一下在求next数组时，为什么要求 $P[0] \cdots P[k-1]$ 这个子串中最大相同前后缀。其实是因为 $P[k]$ 和 $P[q]$ 失配，导致之前的最长前后缀必然要缩小来看有没有可能在之后能够使得 $P[k]$ 和 $P[q]$ 匹配。因此，我们其实要减少的是 $P[q-k] \cdots P[q-1]$ ，看看在其之间有没有一个位置 $j$ 使得 $P[j] \cdots P[q-1]$ 和 $P[0] \cdots P[k'-1]$ 为新的最长前后缀并且 $P[k']$ 和 $P[q]$ 能够匹配。但是因为 $P[q-k] \cdots P[q-1]$ 与 $P[0] \cdots P[k-1]$ 相同（之前的最长前后缀），因此上面说的对 $P[0] \cdots P[k-1]$ 字串作用得到的结果一样，所以我们要求 $P[0] \cdots P[k-1]$ 这个子串中最大相同前后缀，其实背后的本质是求 $P[q-k] \cdots P[q-1]$ 这个子串中最大相同前后缀。

代码如下：

```
public class Solution {
    public int strStr(String haystack, String needle) {
        if(haystack == null || needle == null || needle.length()
> haystack.length()){
            return -1;
        }

        if(needle.length() == 0){
            return 0;
        }

        // int res = -1;
        // for(int i = 0; i < haystack.length(); i++){
        //     if(haystack.length() - i >= needle.length() && ha
ystack.charAt(i) == needle.charAt(0)){
        //         res = i;
        //         int pos = i + 1;
        //         for(int j = 1; j < needle.length(); j++){
        //             if(haystack.charAt(pos++) != needle.charA
t(j)){
        //                 res = -1;
        //                 break;
        //             }
        //         }
        //     }
        // }

        // if(res != -1){
        //     return res;
        // }
```

```
//      }
// }

// return res;

//KMP
int n = haystack.length();
int m = needle.length();
char[] orig = haystack.toCharArray();
char[] target = needle.toCharArray();
int[] next = getNextArray(target);
int q, p;
for(q = 0, p = 0; q < n; q++){
    while(p > 0 && orig[q] != target[p]){
        p = next[p - 1];
    }

    if(orig[q] == target[p]){
        p++;
    }

    if(p == m){
        return q - m + 1;
    }
}

return -1;
}

private int[] getNextArray(char[] target){
    int len = target.length;
    int[] next = new int[len];
    next[0] = 0;
    int k = 0;

    for(int i = 1; i < len; i++){
        while(k > 0 && target[i] != target[k]){
            k = next[k - 1];
        }
    }
}
```

```
        if(target[i] == target[k]){
            k++;
        }

        next[i] = k;
    }

    return next;
}
```

# Multiply Strings 43 (LeetCode)

## Question

Given two numbers represented as strings, return multiplication of the numbers as a string.

Note:

The numbers can be arbitrarily large and are non-negative.

Converting the input string to integer is NOT allowed.

You should NOT use internal library such as BigInteger.

## Solution

创建一个 $\text{len1} + \text{len2}$ 长度的数组用于存放结果，**str1**从最后往前每次取1位(*i*)，和**str2**的每一位(*j*)相乘，结果存在结果数组的 $i+j+1$ 位上。有几点要注意：

1. **str1**的一位和**str2**的每一位相乘后，结果要加上数组在该位上保存的之前的结果和前一位的进位
2. 最高位有可能还需要进位
3. 结果数组是按最长结果开的，从后往前填可能并没有填满，所以要找到第一个不为0的数（即将开头的0都去掉）

代码如下：

```
public class Solution {
    public String multiply(String num1, String num2) {
        if(num1 == null || num2 == null || num1.length() == 0 ||
        num2.length() == 0){
            return "";
        }

        if(num1.equals("0") || num2.equals("0")){
```

```
        return "0";
    }

    int len1 = num1.length();
    int len2 = num2.length();
    int len3 = len1 + len2;
    int[] num = new int[len3];
    int i, j, product, carrier;
    for(i = len1 - 1; i >= 0; i--){
        product = 0;
        carrier = 0;
        for(j = len2 - 1; j >= 0; j--){
            //str1的一位和str2的每一位相乘，结果加上数组在该位之前的结果
            //再加上前一位的进位
            product = carrier + num[i + j + 1] + Character.getNumericValue(num1.charAt(i)) * Character.getNumericValue(num2.charAt(j));
            carrier = product / 10;
            num[i + j + 1] = product % 10;
        }
        //考虑最高位还有进位
        num[i + j + 1] = carrier;
    }

    StringBuilder sb = new StringBuilder();
    i = 0;
    while(i < len3 - 1 && num[i] == 0){
        i++;
    }
    while(i < len3){
        sb.append(num[i++]);
    }

    return sb.toString();
}
}
```

# Enumeration

## Digit Counts 3

### Question

Count the number of k's between 0 and n. k can be 0 - 9.

Example

if  $n = 12$ ,  $k = 1$  in

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

we have FIVE 1's (1, 10, 11, 12)

### Solution

方法一：Brute Force, 0到n个数挨个算过去。最大的问题就是效率，当n非常大时，就需要很长的运行时间。

方法二：参考<http://www.hawstein.com/posts/20.4.html>分析一下会有如下结论

当某一位的数字小于i时，那么该位出现i的次数为：更高位数字 $\times$ 当前位数  
当某一位的数字等于i时，那么该位出现i的次数为：更高位数字 $\times$ 当前位数+低位数字+1  
当某一位的数字大于i时，那么该位出现i的次数为：(更高位数字+1) $\times$ 当前位数

假设一个5位数 $N=abcde$ ，我们现在来考虑百位上出现2的次数，即，从0到abcde的数中，有多少个数的百位上是2。分析完它，就可以用同样的方法去计算个位，十位，千位，万位等各个位上出现2的次数。

当百位c为0时，比如说12013，0到12013中哪些数的百位会出现2？我们从小的数起，200~299, 1200~1299, 2200~2299, ..., 11200~11299, 也就是固定低3位为200~299，然后高位依次从0到11，共12个。再往下12200~12299已经大于12013，因此不再往下。所以，当百位为0时，百位出现2的次数只由更高位决定，等于更高位数字 $(12) \times$ 当前位数 $(100) = 1200$ 个。

当百位c为1时，比如说12113。分析同上，并且和上面的情况一模一样。最大也只能到11200~11299，所以百位出现2的次数也是1200个。

上面两步综合起来，可以得到以下结论：

当某一位的数字小于2时，那么该位出现2的次数为：更高位数字 $\times$ 当前位数 当百位 $c$ 为2时，比如说12213。那么，我们还是有200~299, 1200~1299, 2200~2299, ..., 11200~11299这1200个数，他们的百位为2。但同时，还有一部分12200~12213，共14个(低位数字+1)。所以，当百位数字为2时，百位出现2的次数既受高位影响也受低位影响，结论如下：

当某一位的数字等于2时，那么该位出现2的次数为：更高位数字 $\times$ 当前位数+低位数字+1 当百位 $c$ 大于2时，比如说12313，那么固定低3位为200~299，高位依次可以从0到12，这一次就把12200~12299也包含了，同时也没低位什么事情。因此出现2的次数是：(更高位数字+1) $\times$ 当前位数。结论如下：

当某一位的数字大于2时，那么该位出现2的次数为：(更高位数字+1) $\times$ 当前位数

注意：当 $k$ 为0时，情况特殊，和之前的计算公式不一样，需要单独讨论。

代码如下：

```
class Solution {
    /*
     * param k : As description.
     * param n : As description.
     * return: An integer denote the count of digit k in 1..n
     */
    public int digitCounts(int k, int n) {
        // write your code here
        if(k < 0 || k > 9){
            return 0;
        }

        if(n < 0 || k > n){
            return 0;
        }

        //brute force
        // int count = 0;
        // for(int i = 0; i <= n; i++){
        //     if(i == k){
        //         count++;
        //     }
        // }
```



```
//      continue;
//    }

//    int curt = i;
//    while(curt != 0){
//        if(curt % 10 == k){
//            count++;
//        }
//        curt /= 10;
//    }
// }
// return count;

//analytic version
int result = 0;
int base = 1;
while (n/base > 0) {
    //当前位
    int cur = (n/base)%10;
    //低位
    int low = n - (n/base) * base;
    //高位
    int high = n/(base * 10);

    if (cur == k) {
        //考虑k=0时的特殊情况
        if(k == 0){
            result += (high - 1) * base + low + 1;
        }else{
            result += high * base + low + 1;
        }
    } else if (cur < k) {
        result += high * base;
    } else {
        //考虑k=0时的特殊情况
        if(k == 0){
            result += high * base;
        }else{
            result += (high + 1) * base;
        }
    }
}
```

```
        }
        base *= 10;
    }
    //当k!=0时，都是从最高位数值等于k时开始数起，k=2时从百位为2即200
    开始数起；但是当k为0时，开始的最高位不能为0，所以上面的方法都是从高一位为1开
    始数的，因此在考察个位的情况时，没有考虑个位数是0但是没有高位（即就是数字0）
    的合法情况，所以要+1
    return k == 0? result + 1 : result;
}
};
```

# Longest Common Prefix 78

## Question

Given k strings, find the longest common prefix (LCP).

Example

For strings "ABCD", "ABEF" and "ACEF", the LCP is "A"

For strings "ABCDEFG", "ABCEFG" and "ABCEFA", the LCP is "ABC"

## Solution

这道题就是依次比较每一个字符串，找到所有字符串最长的公共字符串。可以先假设第一个字符串为最长公共字符串，然后用它和其他字符串逐位比较，找到最长的公共字符串，并更新最长公共字符串，直到和所有字符串比较过为止。在这个过程中吗，只要有任何两个字符串的公共字符串长度为0，即返回""。

代码如下：

```
public class Solution {  
    /**  
     * @param strs: A list of strings  
     * @return: The longest common prefix  
     */  
    public String longestCommonPrefix(String[] strs) {  
        // write your code here  
        if(strs == null || strs.length == 0){  
            return "";  
        }  
  
        String lcp = strs[0];  
        for(int i = 0; i < strs.length; i++){  
            int j = 0;  
            while(j < strs[i].length() && j < lcp.length() && st  
rs[i].charAt(j) == lcp.charAt(j)){  
                j++;  
            }  
            if(j == 0){  
                return "";  
            }  
            lcp = lcp.substring(0, j);  
        }  
  
        return lcp;  
    }  
}
```

# Longest Words 133

## Question

Given a dictionary, find all of the longest words in the dictionary.

Example

Given

```
{ "dog", "google", "facebook", "internationalization", "blabla" }
```

the longest words are(is) ["internationalization"].

Given

```
{ "like", "love", "hate", "yes" }
```

the longest words are ["like", "love", "hate"].

Challenge

It's easy to solve it in two passes, can you do it in one pass?

## Solution

一遍法：遍历数组元素，如果当前元素长度大于之前最长元素长度，则将答案列表清空，将当前元素作为答案加入，同时更新最长元素长度；如果当前元素长度等于之前最长元素长度，则直接将当前元素作为答案加入；如果当前元素长度小于之前最长元素长度，则直接跳过。

两遍法：扫一遍求最长元素长度，再扫一遍找到所有长度为最长长度的元素。

代码如下：

```
class Solution {
    /**
     * @param dictionary: an array of strings
     * @return: an arraylist of strings
     */
    ArrayList<String> longestWords(String[] dictionary) {
        // write your code here
        //one pass
        ArrayList<String> res = new ArrayList<String>();
        if(dictionary == null || dictionary.length == 0){
            return res;
        }

        int max = 0;
        for(String s : dictionary){
            if(s.length() > max){
                max = s.length();
                res.clear();
                res.add(s);
            } else if(s.length() == max){
                res.add(s);
            }
        }

        return res;
    }
};
```

# Fizz Buzz 9

## Question

Given number n. Print number from 1 to n. But:

when number is divided by 3, print "fizz".

when number is divided by 5, print "buzz".

when number is divided by both 3 and 5, print "fizz buzz".

Example

If n = 15, you should return:

```
[ "1", "2", "fizz", "4", "buzz", "fizz", "7", "8", "fizz", "buzz", "11", "fizz", "13", "14", "fizz buzz" ]
```

## Solution

看能否被15，3，5整除即可。

代码如下：

```
class Solution {  
    /**  
     * param n: As description.  
     * return: A list of strings.  
     */  
    public ArrayList<String> fizzBuzz(int n) {  
        ArrayList<String> results = new ArrayList<String>();  
        for (int i = 1; i <= n; i++) {  
            if (i % 15 == 0) {  
                results.add("fizz buzz");  
            } else if (i % 5 == 0) {  
                results.add("buzz");  
            } else if (i % 3 == 0) {  
                results.add("fizz");  
            } else {  
                results.add(String.valueOf(i));  
            }  
        }  
        return results;  
    }  
}
```



# Mathematics

# Ugly Number 517

## Question

Write a program to check whether a given number is an ugly number`.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5. For example, 6, 8 are ugly while 14 is not ugly since it includes another prime factor 7.

Notice

Note that 1 is typically treated as an ugly number.

Example

Given num = 8 return true

Given num = 14 return false

## Solution

判断能否被2，3，5除尽。

代码如下：

```
public class Solution {  
    /**  
     * @param num an integer  
     * @return true if num is an ugly number or false  
     */  
    public boolean isUgly(int num) {  
        // Write your code here  
        if(num <= 0){  
            return false;  
        }  
  
        if(num == 1){  
            return true;  
        }  
  
        while(num >= 2 && num % 2 == 0){  
            num /= 2;  
        }  
  
        while(num >= 3 && num % 3 == 0){  
            num /= 3;  
        }  
  
        while(num >= 5 && num % 5 == 0){  
            num /= 5;  
        }  
  
        return num == 1;  
    }  
}
```

# ZigZag Conversion (LeetCode 6)

## Question

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

P A H N A P L S I I G Y I R

And then read line by line: "PAHNAPLSIIGYIR"

Write the code that will take a string and make this conversion given a number of rows:

```
string convert(string text, int numRows);
```

`convert("PAYPALISHIRING", 3)` should return "PAHNAPLSIIGYIR".

## Solution

这是一道数学题，就是找规律。

可以发现，字符在主列和主列之间。主列之间间隔的距离为  $step = 2 (numRows - 1)$ 。然后除了第一行和最后一行外，其他每行都要加中间的数。对于第  $i$  行，其主列和下一个主列之间元素的间距为  $interval = step - i * 2$ ，即如果当前主列为  $j$ ，则下一个主列之间的元素插入位置为  $j + interval$ 。逐行遍历直到结束。

代码如下：

```
public class Solution {
    public String convert(String s, int numRows) {
        if(s.length() == 0 || numRows == 1 || numRows >= s.length()) {
            return s;
        }

        char[] res = new char[s.length()];
        int step = 2 * (numRows - 1);
        int count = 0;

        for(int i = 0; i < numRows; i++){
            int interval = step - 2 * i;
            for(int j = i; j < s.length(); j = j + step){
                res[count++] = s.charAt(j);
                //第一行 (interval == step) 和最后一行 (interval == 0) 不用加之间的数
                if(interval < step && interval > 0 && j + interval < s.length() && count < s.length()){
                    res[count++] = s.charAt(j + interval);
                }
            }
        }

        return new String(res);
    }
}
```

# Palindrome Number (Leetcode 9)

## Question

Determine whether an integer is a palindrome. Do this without extra space.

click to show spoilers.

Some hints: Could negative integers be palindromes? (ie, -1)

If you are thinking of converting the integer to string, note the restriction of using extra space.

You could also try reversing an integer. However, if you have solved the problem "Reverse Integer", you know that the reversed integer might overflow. How would you handle such case?

There is a more generic way of solving this problem.

## Solution

又是一道数学题。判断是否是回文数字，只要将数字反转即可（和[string to integer](#)类似）。但是这里其实不用考虑溢出的问题，因为给的是一个integer，如果是回文，则反转之后一定还是原来的integer，如果溢出则一定不会是回文。负数也不是回文。

代码如下：

```
public class Solution {
    public boolean isPalindrome(int x) {
        if(x < 0){
            return false;
        }

        int reverse = 0;
        int temp = x;
        while(temp != 0){
            reverse = reverse * 10 + temp % 10;
            temp /= 10;
        }

        return reverse == x;
    }
}
```

# Count Primes

## Question

Count the number of prime numbers less than a non-negative number,  $n$ .

## Solution

这道题首先想到的方法是从1到 $n-1$ 依次验证是否为prime。但是这样的时间复杂度为 $O(n^2)$ 。

如果使用埃拉托斯特尼筛法（Sieve of Eratosthenes），则可以将时间复杂度降低到 $O(n \log \log n)$ 。具体方法如下：

	2	3	4	5	6	7	8	9	10	Prime numbers
11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	
31	32	33	34	35	36	37	38	39	40	
41	42	43	44	45	46	47	48	49	50	
51	52	53	54	55	56	57	58	59	60	
61	62	63	64	65	66	67	68	69	70	
71	72	73	74	75	76	77	78	79	80	
81	82	83	84	85	86	87	88	89	90	
91	92	93	94	95	96	97	98	99	100	
101	102	103	104	105	106	107	108	109	110	
111	112	113	114	115	116	117	118	119	120	

这个算法的过程如上图所示，我们从2开始遍历到根号 $n$ ，先找到第一个质数2，然后将其所有的倍数全部标记出来，然后到下一个质数3，标记其所有倍数，一次类推，直到根号 $n$ ，此时数组中未被标记的数字就是质数。之所以不用考虑比自己小的倍数（比如3从3倍开始而不用考虑2倍）是因为自己已经被作为小的数的倍数考虑过（即2的3倍），因此每一个数都从自身倍数开始直到 $n$ 。

代码如下：



```
public class Solution {
    public int countPrimes(int n) {
        boolean[] isPrime = new boolean[n];
        for (int i = 0; i < n; i++) {
            isPrime[i] = true;
        }

        for (int i = 2; i * i < n; i++) {
            if (!isPrime[i]) {
                continue;
            }
            for (int j = i * i; j < n; j += i) {
                isPrime[j] = false;
            }
        }

        int count = 0;
        for (int i = 2; i < n; i++) {
            if (isPrime[i]) {
                count++;
            }
        }

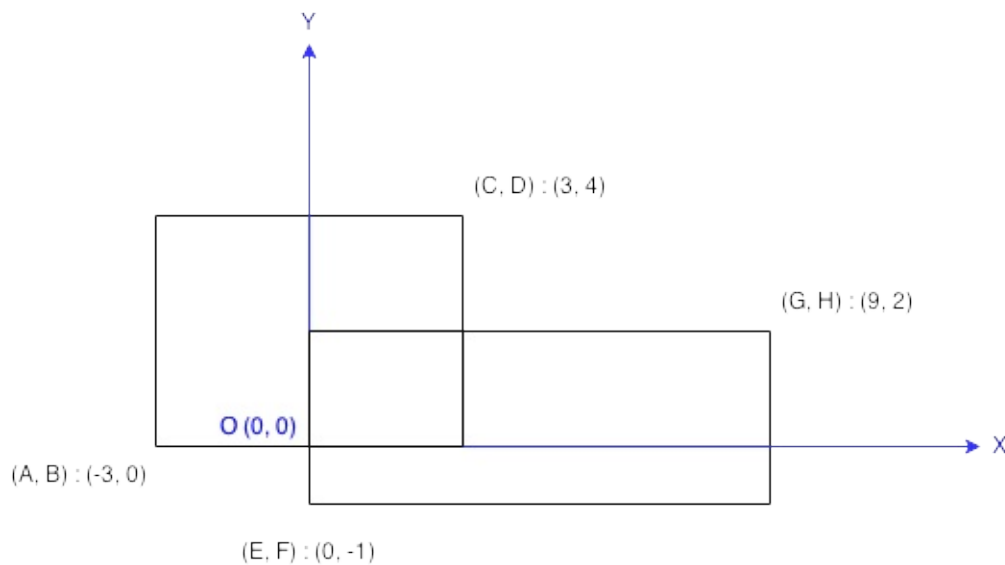
        return count;
    }
}
```

## 223. Rectangle Area

### Question

Find the total area covered by two rectilinear rectangles in a 2D plane.

Each rectangle is defined by its bottom left corner and top right corner as shown in the figure.



Assume that the total area is never beyond the maximum possible value of int.

### Solution

这道题最关键的是要找出重叠的部分，两个矩形的面积和-重叠部分的面积就是答案。

观察重叠部分，肯定都是在中间，需要找出重叠部分上下左右的边。重叠部分左边界肯定是两个矩形左边界的较大者，同理，重叠部分右边界肯定是两个矩形右边界的较小者。这里可以用一个小技巧，即取右边界时，和刚才取的左边界比较，取两者中的较大者为右边界，这样就把两个矩形没有重叠的情况也包括进来了（没有重叠的话在这里重叠部分的左右边界会相同，也就是宽为0，面积自然就是0），不用单独讨论了。以此类推，可以求出重叠部分的上下边界。

代码如下：

```
public class Solution {  
    public int computeArea(int A, int B, int C, int D, int E, int F, int G, int H) {  
        int left = Math.max(A, E);  
        int right = Math.max(left, Math.min(C, G));  
        int bottom = Math.max(B, F);  
        int top = Math.max(bottom, Math.min(D, H));  
  
        return (C - A) * (D - B) + (G - E) * (H - F) - (right - left) * (top - bottom);  
    }  
}
```

**Others**

# Google

# Zigzag Iterator 540

## Question

Given two 1d vectors, implement an iterator to return their elements alternately.

Example

Given two 1d vectors:

v1 = [1, 2]

v2 = [3, 4, 5, 6]

By calling next repeatedly until hasNext returns false, the order of elements returned by next should be: [1, 3, 2, 4, 5, 6].

## Solution

这道题可以直接调用list的iterator来实现，只要用一个计数来确定应该用哪个iteraor即可。当然也可以自己用两个index来实现，但是iterator在内存中占用空间更小，并且更方便操作。

代码如下：

```
public class ZigzagIterator {
    /**
     * @param v1 v2 two 1d vectors
     */
    //用iterator更省空间
    Iterator<Integer> it1;
    Iterator<Integer> it2;
    int count;
    public ZigzagIterator(List<Integer> v1, List<Integer> v2) {
        // initialize your data structure here.
        it1 = v1.iterator();
        it2 = v2.iterator();
        count = 0;
    }
}
```

```
    }

    public int next() {
        // Write your code here
        count++;
        if((count % 2 == 1 && it1.hasNext()) || !it2.hasNext()){
            return it1.next();
        }else if((count % 2 == 0 && it2.hasNext()) || !it1.hasNext()){
            return it2.next();
        }
        return -1;
    }

    public boolean hasNext() {
        // Write your code here
        return it1.hasNext() || it2.hasNext();
    }
}

/**
 * Your ZigzagIterator object will be instantiated and called as
 * such:
 * ZigzagIterator solution = new ZigzagIterator(v1, v2);
 * while (solution.hasNext()) result.add(solution.next());
 * Output result
 */
```

# Zigzag Iterator II 541

## Question

Follow up Zigzag Iterator: What if you are given  $k$  1d vectors? How well can your code be extended to such cases? The "Zigzag" order is not clearly defined and is ambiguous for  $k > 2$  cases. If "Zigzag" does not look right to you, replace "Zigzag" with "Cyclic".

Example

Given  $k = 3$  1d vectors:

[1,2,3]

[4,5,6,7]

[8,9]

Return [1,4,8,2,5,9,3,6,7].

## Solution

和I类似，只是用一个list来存 $k$ 个iterator（如果iterator中没有元素则不用加入）。用一个 $\text{count} \% \text{list\_size}$ 来确定应该用哪个iterator。若其中一个iterator里面的数已经被取完（即该iterator的 $\text{hasNext()}$ 为false），则将其从list中移除，并将count设置为 $\text{count} \% \text{new list\_size}$ （如果 $\text{new list\_size} \neq 0$ ）。

代码如下：

```
public class ZigzagIterator2 {
    /**
     * @param vecs a list of 1d vectors
     */
    //用iterator更省空间
    private ArrayList<Iterator<Integer>> vec;
    private int count;
    public ZigzagIterator2(ArrayList<ArrayList<Integer>> vecs) {
```



```

        // initialize your data structure here.
        vec = new ArrayList<Iterator<Integer>>();
        for(ArrayList<Integer> list : vecs){
            if(list.size() > 0){
                vec.add(list.iterator());
            }
        }
        count = 0;
    }

    public int next() {
        // Write your code here
        int res = vec.get(count).next();
        if(vec.get(count).hasNext()){
            count = (count + 1) % vec.size();
        }else{
            vec.remove(count);
            if(vec.size() > 0){
                count %= vec.size();
            }
        }
        return res;
    }

    public boolean hasNext() {
        // Write your code here
        return vec.size() > 0;
    }
}

/**
 * Your ZigzagIterator2 object will be instantiated and called a
s such:
 * ZigzagIterator2 solution = new ZigzagIterator2(vecs);
 * while (solution.hasNext()) result.add(solution.next());
 * Output result
 */

```



# Insert Interval 30

## Question

Given a non-overlapping interval list which is sorted by start point.

Insert a new interval into it, make sure the list is still in order and non-overlapping (merge intervals if necessary).

Example

Insert [2, 5] into [[1,2], [5,9]], we get [[1,9]].

Insert [3, 4] into [[1,2], [5,9]], we get [[1,2], [3,4], [5,9]].

## Solution

用pos记录newInterval应该插入的位置。顺序遍历intervals中的元素，若当前interval的end比newInterval的start还小，则将当前interval加入答案，同时pos+1；若比newInterval大，则直接加入答案；若有overlap，则需要merge，newInterval的start取两者间小的，end取两者间大的。最后在pos的位置插入newInterval即可。

代码如下：

```
/**
 * Definition of Interval:
 * public classs Interval {
 *     int start, end;
 *     Interval(int start, int end) {
 *         this.start = start;
 *         this.end = end;
 *     }
 */

class Solution {
    /**
     * Insert newInterval into intervals.
```

```
* @param intervals: Sorted interval list.
* @param newInterval: A new interval.
* @return: A new sorted interval list.
*/
public ArrayList<Interval> insert(ArrayList<Interval> intervals, Interval newInterval) {
    ArrayList<Interval> result = new ArrayList<Interval>();
    // write your code here
    if(newInterval == null){
        result = intervals;
        return result;
    }

    if(intervals == null || intervals.size() == 0){
        result.add(newInterval);
        return result;
    }

    int pos = 0;
    for(Interval interval : intervals){
        if(interval.end < newInterval.start){
            result.add(interval);
            pos++;
        }else if(interval.start > newInterval.end){
            result.add(interval);
        }else{
            newInterval.start = Math.min(interval.start, newInterval.start);
            newInterval.end = Math.max(interval.end, newInterval.end);
        }
    }

    result.add(pos, newInterval);

    return result;
}
```



# FaceBook

# Add Binary 408

## Question

Given two binary strings, return their sum (also a binary string).

Example

a = 11

b = 1

Return 100

## Solution

方法一：将两个数转换成二进制整数，然后用位运算进行相加，再将所得结果用二进制string表示。

方法二：将长的那个数放在前，短的数放在后。用carriers记录进位。然后从后往前将两个数对应的位置上的数相加上carriers，将结果%2加入res之前，同时用结果/2来更新carriers。当短的数到头之后，则只要将长的数的位上的数和carriers相加即可。最后当长的数也到头之后，若carriers为1，则在res前加"1"即可。

代码如下：

方法一：

```
public class Solution {  
    /**  
     * @param a a number  
     * @param b a number  
     * @return the result  
     */  
    public String addBinary(String a, String b) {  
        // Write your code here  
        int c = Integer.parseInt(a, 2);  
        int d = Integer.parseInt(b, 2);  
        while(d != 0){  
            int _c = c ^ d;  
            int _d = (c & d) << 1;  
            c = _c;  
            d = _d;  
        }  
  
        return Integer.toBinaryString(c);  
    }  
}
```

方法二：



```
public class Solution {
    public String addBinary(String a, String b) {
        if(a.length() < b.length()){
            String tmp = a;
            a = b;
            b = tmp;
        }

        int pa = a.length()-1;
        int pb = b.length()-1;
        int carries = 0;
        String rst = "";

        while(pb >= 0){
            int sum = (int)(a.charAt(pa) - '0') + (int)(b.charAt
(pb) - '0') + carries;
            rst = String.valueOf(sum % 2) + rst;
            carries = sum / 2;
            pa--;
            pb--;
        }

        while(pa >= 0){
            int sum = (int)(a.charAt(pa) - '0') + carries;
            rst = String.valueOf(sum % 2) + rst;
            carries = sum / 2;
            pa--;
        }

        if (carries == 1)
            rst = "1" + rst;
        return rst;
    }
}
```

# Count and Say 420

## Question

The count-and-say sequence is the sequence of integers beginning as follows:

1, 11, 21, 1211, 111221, ...

1 is read off as "one 1" or 11.

11 is read off as "two 1s" or 21.

21 is read off as "one 2, then one 1" or 1211.

Given an integer  $n$ , generate the  $n$ th sequence.

Notice

The sequence of integers will be represented as a string.

Example

Given  $n = 5$ , return "111221".

## Solution

遍历上一个字符串，记下当前字符连续重复出现的次数，将次数+当前元素加入本次答案。

代码如下：

```
public class Solution {  
    /**  
     * @param n the nth  
     * @return the nth sequence  
     */  
    public String countAndSay(int n) {  
        // Write your code here  
        if(n < 1){  
            return "";  
        }  
  
        String s = "1";  
        for(int i = 1; i < n; i++){  
            String next = "";  
            int j = 0;  
            while(j < s.length()){  
                int start = j;  
                char curt = s.charAt(j++);  
                while(j < s.length() && s.charAt(j) == curt){  
                    j++;  
                }  
                int num = j - start;  
                next += num + Character.toString(curt);  
            }  
            s = next;  
        }  
  
        return s;  
    }  
}
```

**No Tag**

# Left Pad 524

## Question

You know what, left pad is javascript package and referenced by React: Github link

One day his author unpublished it, then a lot of javascript projects in the world broken.

You can see from github it's only 11 lines.

Your job is to implement the left pad function. If you do not know what left pad does, see examples below and guess.

Example

```
leftpad("foo", 5)
```

```
  |  |  " foo"
```

```
leftpad("foobar", 6)
```

```
  |  |  "foobar"
```

```
leftpad("1", 2, "0")
```

```
  |  |  "01"
```

## Solution

非常简单，直接判断要在原始string之前添加多少个其他字符即可。

代码如下：

```
public class StringUtils {
    /**
     * @param originalStr the string we want to append to with s
    paces
     * @param size the target length of the string
     * @return a string
     */
    static public String leftPad(String originalStr, int size) {
        // Write your code here
        if(originalStr.length() >= size){
            return originalStr;
        }else{
            char[] c = new char[size- originalStr.length()];
            Arrays.fill(c, ' ');
            return new String(c) + originalStr;
        }
    }

    /**
     * @param originalStr the string we want to append to
     * @param size the target length of the string
     * @param padChar the character to pad to the left side of t
    he string
     * @return a string
     */
    static public String leftPad(String originalStr, int size, c
    har padChar) {
        // Write your code here
        if(originalStr.length() >= size){
            return originalStr;
        }else{
            char[] c = new char[size- originalStr.length()];
            Arrays.fill(c, padChar);
            return new String(c) + originalStr;
        }
    }
}
```



# Reverse Integer 413

## Question

Reverse digits of an integer. Returns 0 when the reversed integer overflows (signed 32-bit integer).

Example

Given  $x = 123$ , return 321

Given  $x = -123$ , return -321

## Solution

`sum`表示之前取的数字的组合，只要不断地取当前数字的最低位，加到`sum` 10中，直到当前数字为0为止。其中有一点需要注意，在将当前最低位加入到`sum` 10后，要再将新的结果/10看能不能得到原来的`sum`，若得不到原来的数，则说明已经溢出（ $-2^{32} \sim 2^{32}$ 。 $-35 \% 10 = -5$ 。

代码如下：



```
public class Solution {  
    /**  
     * @param n the integer to be reversed  
     * @return the reversed integer  
     */  
    public int reverseInteger(int n) {  
        // Write your code here  
        int sum = 0;  
        while(n != 0){  
            int temp = sum * 10 + n % 10;  
            n /= 10;  
            //若*10之后的数再/10得不到原来的数，则说明已经溢出（-2^32~2  
^32）  
            if(temp / 10 != sum){  
                sum = 0;  
                break;  
            }  
            sum = temp;  
        }  
        return sum;  
    }  
}
```

# Cosine Similarity 445

## Question

Cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them. The cosine of  $0^\circ$  is 1, and it is less than 1 for any other angle.

See wiki: [Cosine Similarity](#)

Here is the formula:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

Given two vectors A and B with the same size, calculate the cosine similarity.

Return 2.0000 if cosine similarity is invalid (for example A = [0] and B = [0]).

Example

Given A = [1, 2, 3], B = [2, 3, 4].

Return 0.9926.

Given A = [0], B = [0].

Return 2.0000

## Solution

根据公式写即可。

代码如下：

```
class Solution {
    /**
     * @param A: An integer array.
     * @param B: An integer array.
     * @return: Cosine similarity.
     */
    public double cosineSimilarity(int[] A, int[] B) {
        // write your code here
        if(A == null || A.length == 0 || B == null || B.length =
= 0){
            return 2.0;
        }

        double sumA = 0;
        double sumB = 0;
        for(int i = 0; i < A.length; i++){
            sumA += A[i] * A[i];
            sumB += B[i] * B[i];
        }

        if(sumA == 0 || sumB == 0){
            return 2.0;
        }

        double crossSum = 0;
        for(int i = 0; i < A.length; i++){
            crossSum += A[i] * B[i];
        }

        return crossSum / (Math.sqrt(sumA) * Math.sqrt(sumB));
    }
}
```

# Word Count (Map Reduce) 499

## Question

Using map reduce to count word frequency.

Example

chunk1: "Google Bye GoodBye Hadoop code"

chunk2: "lintcode code Bye"

Get MapReduce result:

```
Bye: 2  
  
GoodBye: 1  
  
Google: 1  
  
Hadoop: 1  
  
code: 2  
  
lintcode: 1
```

## Solution

MapReduce的map和reduce基本操作。

代码如下：

```

/**
 * Definition of OutputCollector:
 * class OutputCollector<K, V> {
 *     public void collect(K key, V value);
 *     // Adds a key/value pair to the output buffer
 * }
 */
public class WordCount {

    public static class Map {
        public void map(String key, String value, OutputCollector<String, Integer> output) {
            // Write your code here
            // Output the results into output buffer.
            // Ps. output.collect(String key, int value);
            StringTokenizer iter = new StringTokenizer(value);
            while(iter.hasMoreTokens()){
                String s = iter.nextToken();
                output.collect(s, 1);
            }
        }
    }

    public static class Reduce {
        public void reduce(String key, Iterator<Integer> values,
                           OutputCollector<String, Integer> output) {
            // Write your code here
            // Output the results into output buffer.
            // Ps. output.collect(String key, int value);
            int sum = 0;
            while(values.hasNext()){
                sum += values.next();
            }
            output.collect(key, sum);
        }
    }
}

```



# Product of Array Exclude Itself 50

## Question

Given an integers array A.

Define  $B[i] = A[0] \dots A[i-1] A[i+1] \dots * A[n-1]$ , calculate B WITHOUT divide operation.

Example

For A = [1, 2, 3], return [6, 3, 2].

## Solution

有点动态规划的思想。

首先从后往前遍历数组，用  $f[i]$  保存  $i$  到  $\text{end}$  所有元素的乘积。

然后从前往后遍历数组，对于当前位  $i$ ，其值的计算公式为  $\text{start} \sim i-1$  的元素的乘积 \*  $f[i]$ 。  $\text{start} \sim i-1$  的乘积在  $i-1$  位置时已经可以得到。

代码如下：

```
public class Solution {
    /**
     * @param A: Given an integers array A
     * @return: A Long array B and B[i]= A[0] * ... * A[i-1] * A
     [i+1] * ... * A[n-1]
     */
    public ArrayList<Long> productExcludeItself(ArrayList<Intege
r> A) {
        // write your code
        //DP
        ArrayList<Long> res = new ArrayList<Long>();
        if(A == null || A.size() == 0){
            return res;
        }
    }
```

```
int n = A.size();
//f[i]:product from i to end
long[] f = new long[n];
f[n - 1] = A.get(n - 1);
for(int i = n - 2; i >= 0; i--){
    f[i] = A.get(i) * f[i + 1];
}

//从开头到当前元素之前的元素之积
long now = 1;
//从开头到当前元素之积
long temp = 1;
for(int i = 0; i < n; i++){
    now = temp;
    if(i + 1 < n){
        res.add(now * f[i + 1]);
    }else{
        res.add(now);
    }
    temp = now * A.get(i);
}

return res;
}
}
```



# Shape Factory 497

## Question

Factory is a design pattern in common usage. Implement a ShapeFactory that can generate correct shape.

You can assume that we have only tree different shapes: Triangle, Square and Rectangle.

Example

```
ShapeFactory sf = new ShapeFactory();
```

```
Shape shape = sf.getShape("Square");
```

```
shape.draw();
```



```
shape = sf.getShape("Triangle");
```

```
shape.draw();
```



```
shape = sf.getShape("Rectangle");
```

```
shape.draw();
```



## Solution

子类函数的重载（override）。注意转义字符。

代码如下：

```
/**
 * Your object will be instantiated and called as such:
 * ShapeFactory sf = new ShapeFactory();
 * Shape shape = sf.getShape(shapeType);
 * shape.draw();
 */
interface Shape {
    void draw();
}

class Rectangle implements Shape {
    // Write your code here
    public void draw(){
        System.out.println(" ---- ");
        System.out.println("|    |");
        System.out.println(" ---- ");
    }
}

class Square implements Shape {
    // Write your code here
    public void draw(){
        System.out.println(" ---- ");
        System.out.println("|    |");
        System.out.println("|    |");
        System.out.println(" ---- ");
    }
}

class Triangle implements Shape {
    // Write your code here
    //注意转义字符
    public void draw(){
        System.out.println(" /\\"");
    }
}
```

```
        System.out.println(" /  \\");
        System.out.println("/____\\");
    }
}

public class ShapeFactory {
    /**
     * @param shapeType a string
     * @return Get object of type Shape
     */
    public Shape getShape(String shapeType) {
        // Write your code here
        if(shapeType.equals("Square")){
            return new Square();
        }else if(shapeType.equals("Triangle")){
            return new Triangle();
        }else if(shapeType.equals("Rectangle")){
            return new Rectangle();
        }else{
            return null;
        }
    }
}
```

# Toy Factory 496

## Question

Factory is a design pattern in common usage. Please implement a ToyFactory which can generate proper toy based on the given type.

Example

```
ToyFactory tf = ToyFactory();
```

```
Toy toy = tf.getToy('Dog');
```

```
toy.talk();
```

```
|| Wow
```

```
toy = tf.getToy('Cat');
```

```
toy.talk();
```

```
|| Meow
```

## Solution

和shape factory一样，重载函数。

代码如下：

```
/**
 * Your object will be instantiated and called as such:
 * ToyFactory tf = new ToyFactory();
 * Toy toy = tf.getToy(type);
 * toy.talk();
 */
interface Toy {
    void talk();
}

class Dog implements Toy {
    // Write your code here
    public void talk(){
        System.out.println("Wow");
    }
}

class Cat implements Toy {
    // Write your code here
    public void talk(){
        System.out.println("Meow");
    }
}

public class ToyFactory {
    /**
     * @param type a string
     * @return Get object of the type
     */
    public Toy getToy(String type) {
        // Write your code here
        if(type.equals("Dog")){
            return new Dog();
        }else if(type.equals("Cat")){
            return new Cat();
        }
        return null;
    }
}
```



# Singleton 204

## Question

Singleton is a most widely used design pattern. If a class has and only has one instance at every moment, we call this design as singleton. For example, for class Mouse (not a animal mouse), we should design it in singleton.

You job is to implement a getInstance method for given class, return the same instance of this class every time you call this method.

## Solution

单例模式，要考虑并发情况。

为了防止多个线程创建多个实例，要在new之前加一个锁，即synchronized。同时为了防止多个线程在判断实例为null之后先后进入synchronized依然创建多个实例，可以在synchronized之后double check一下实例是否为null。更多讨论见下：

### Reference

代码如下：

```
class Solution {  
    /**  
     * @return: The same instance of this class every time  
     */  
    //考察静态变量  
    public static Solution s = null;  
    public static Solution getInstance() {  
        // write your code here  
        if(s == null){  
            synchronized (Solution.class){  
                if(s == null){  
                    s = new Solution();  
                }  
            }  
        }  
  
        return s;  
    }  
};
```



## 其他问题

# Use Dynamic Circulate Array to Implement Deque

## Question

实现Deque的方法：get(), insert from front or end (add or push), delete from front or end (pop or poll), 同时当添加元素数量超过原始数组长度时，数组可以动态扩张。所有方法的时间复杂度为  $O(1)$ 。

## Solution

要在前后两端插入和删除，考虑使用循环数组结构。用front记录数组头（不为null），用end记录数组尾（为null），数组可以插入数组长度-1个元素，用 $end+1 \% size == front$ 来判断队列是否full，用 $end == front$ 来判断对列是否空。当数组为full时还要插入元素则需要将数组扩展为原来的两倍。

同时使用数组动态扩展的方法System.arraycopy()。

Reference:

[Java数组实现可以动态增长的队列](#)

[java队列实现（顺序队列、链式队列、循环队列](#)

代码如下：

```
class Mydequeue<E>{
    int front;
    int end;
    Object[] storage;
    int initialLen = 5;

    public Mydequeue(){
        this.front = this.end = 0;
        storage = new Object[initialLen];
    }
}
```

```
//判断数组是否满
public boolean isFull(){
    if((end + 1) % storage.length == front || (front - 1) %
storage.length == end){
        return true;
    }
    return false;
}

//判断数组是否空
public boolean isEmpty(){
    if(front == end){
        return true;
    }
    return false;
}

//从数组头加入元素
public void enqueueFromFront(E elem){
    if(isFull){
        enLarge();
        System.out.println("队列容量不够，已动态增加数组容量到："+
storage.length);
    }

    front = (front - 1) % storage.length;
    storage[front] = elem;
}

//从数组尾加入元素
public void enqueueFromEnd(E elem){
    if(isFull){
        enLarge();
        System.out.println("队列容量不够，已动态增加数组容量到："+
storage.length);
    }

    storage[end] = elem;
    end = (end + 1) % storage.length;
}

//从数组头删除元素
public Object dequeueFromFront(){
    if(isEmpty()){
```

```

        System.out.println("队列为空，无法出队");
    }

    storage[front] = null;
    front = (front + 1) % storage.length;
}
//从数组尾删除元素
public Object dequeueFromEnd(){
    if(isEmpty()){
        System.out.println("队列为空，无法出队");
    }

    end = (end - 1) % storage.length;
    storage[end] = null;
}
//获取下标为index的元素
public Object get(int index){
    return storage[index];
}
//打印数组
public void printAll(){
    for(Object i : storage){
        System.out.print(i + " ");
    }
    System.out.println();
}
//扩张数组
public void enLarge(){
    //新数组为原数组的两倍
    Object a = new Object[storage.length * 2];
    //当未填充位置在最后时，直接将前面的元素复制给新数组
    if(end == storage.length - 1){
        System.arraycopy(storage, 0, a, 0, storage);
    }else{
        //当未填充位置在end和front之间时，原数组分为两段，将0~end的元素
        //直接复制到新数组，将front和之后的元素复制到新数组的尾部，这样end和front之
        //间又有了可以填充的空间，实现了数组的扩张。最后记得要更新front在新数组中的in
        //dex（end不变）。
        System.arraycopy(storage, 0, a, 0, end + 1);
        System.arraycopy(storage, front, a, a.length - stora

```

```
ge.length + front, storage.length - front);
    front = a.length - storage.length + front;
}

    storage = a;
}
}
```

# Random Generator

## Question

给一个1到3的random generator，构造一个1到n的random generator。

## Solution

用3进制数来求解。首先求出n至少需要几位3进制数来表示，假设需要x位。然后利用1到3的random generator来产生依次产生每一位上的随机数（因为3进制每一位上是0到2，所以要将得到的随机数+1）。再将得到的随机数转化为十进制整数。因为x位3进制的数能表示的十进制的数的范围是 $0 \sim 3^x - 1$ ，因此得到的随机数可能比n大。如果得到的结果大于n，则重新产生随机数。

如果调用这个n random generator很多次，很可能产生很多次重复的随机数。因此，可以将得到的合法3进制的数及其对应的十进制的数保存在一个table里来优化时间。每次产生一个随机数，就和n比较，如果大于n，则重新产生，否则去table寻找其对应的十进制数。如果没有找到，则说明该随机数第一次产生，将该随机数的3进制及其十进制形式保存在table里。

# Array Monotonic

## Question

给一个数组array，判断该数组是否是单调的。如果数字都相同为true。

## Solution

这道题很简单，假设数组单调增或者单调减，然后逐个元素比较判断即可。唯一的trick就是首先要找出该数组是增还是减的假设。当元素个数大于等于3个时，首先比较第一个元素和最后一个元素的大小，如果第一个元素大，则假设数组递减，否则假设数组递增。如果相等，则去判断数组中元素是否全部相等。

conor case：当数组元素小于3个时：如果0个元素，则false，1或者2个元素，则true（2个元素不是增就是减）。

## 待解决

Longest Palindromic Substring 200 中心展开法 $O(n)$ 实现

HashHeap